NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOFTWARE DEFINED NETWORK MONITORING SCHEME USING SPECTRAL GRAPH THEORY AND PHANTOM NODES

by

Jamie L. Johnson

September 2014

Thesis Co-Advisors: Murali Tummala
John McEachen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2014	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
SOFTWARE DEFINED NETWORK MONITORING SCHEME USING SPECTRAL GRAPH THEORY AND PHANTOM NODES

5. FUNDING NUMBERS

6. AUTHOR(S) Jamie L. Johnson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

In this thesis, we propose a new software defined network monitoring scheme that provides the controller with a method to determine network states for the purpose of updating flow rules for network control and management. Network centrality and nodal influence metrics derived from the dual basis concept of the graph theory are used to monitor changes in a network. The proposed scheme uses a phantom node and the concept of a reference node to determine changes in these metrics in order to identify disconnected, congested, underutilized, and attacked nodes. The phantom node establishes a congestion threshold in the dual basis that is used to determine changes in node health and capacity due to network traffic. Multiple phantom nodes are used to produce multiple congestion thresholds for network monitoring. A congestion estimation method is proposed to estimate a node's capacity used when it crosses the congestion threshold. Simulations are used to validate the concept of reference node, identification of node disconnections, congestion, and attacks, and the congestion estimation method.

14. SUBJECT TERMS software defined network, graph theory, phantom node, reference node, network monitoring, network centrality, nodal influence, dual basis, congestion, attack.

15. NUMBER OF PAGES
121

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SOFTWARE DEFINED NETWORK MONITORING SCHEME USING SPECTRAL GRAPH THEORY AND PHANTOM NODES

Jamie L. Johnson
Lieutenant, United States Navy
B.A., University of California San Diego, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2014

Author: Jamie L. Johnson

Approved by: Murali Tummala
Thesis Co-Advisor

John C. McEachen
Thesis Co-Advisor

Ralph C. Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, we propose a new software defined network monitoring scheme that provides the controller with a method to determine network states for the purpose of updating flow rules for network control and management. Network centrality and nodal influence metrics derived from the dual basis concept of the graph theory are used to monitor changes in a network. The proposed scheme uses a phantom node and the concept of a reference node to determine changes in these metrics in order to identify disconnected, congested, underutilized, and attacked nodes. The phantom node establishes a congestion threshold in the dual basis that is used to determine changes in node health and capacity due to network traffic. Multiple phantom nodes are used to produce multiple congestion thresholds for network monitoring. A congestion estimation method is proposed to estimate a node's capacity used when it crosses the congestion threshold. Simulations are used to validate the concept of reference node, identification of node disconnections, congestion, and attacks, and the congestion estimation method.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
- **A. BACKGROUND 1**
- **B. THESIS OBJECTIVES AND APPROACH 2**
- **C. RELATED WORK 3**
- **D. THESIS ORGANIZATION 4**

II. BACKGROUND 5
- **A. SOFTWARE DEFINED NETWORKS 5**
  - **1. Architecture 5**
  - **2. Routing 6**
    - ***a. Packet Matching 7***
    - ***b. Flow Entries 8***
    - ***c. Polling the Switch 9***
- **B. NETWORK CONGESTION AND ATTACKS 10**
- **C. GRAPH THEORY 11**
  - **1. Spectral Graph Theory 13**

III. PROPOSED NETWORK MONITORING SCHEME 17
- **A. DUAL BASIS 17**
  - **1. Orthogonality 18**
  - **2. Eigencentrality Basis and Nodal Basis 19**
    - ***a. Network Centrality 19***
    - ***b. Nodal Influence 21***
- **B. PHANTOM NODES 23**
  - **1. Reference Node 24**
  - **2. Placing the Congestion Threshold 26**
- **C. PROPOSED MONITORING SCHEME 27**
  - **1. Determine Location of Phantom Node 28**
  - **2. Determine Network State 29**
  - **3. Compute Eigenvectors and Eigenvalues 30**
  - **4. Analyze Connectivity 32**
  - **5. Determine Congestion, Underutilization, and Attack 32**
  - **6. Determine Outputs for Controller Action 34**
- **D. CONGESTION ESTIMATION METHOD 35**
  - **1. Algebraic Connectivity Boundary Equations 36**
  - **2. Estimating Congestion for Threshold Crossing 38**

IV. SIMULATION AND RESULTS 41
- **A. METHODOLOGY 41**
- **B. CONNECTIVITY RESULTS 43**
  - **1. Disconnected Nodes Simulation 43**
  - **2. Link Failure Simulation 47**
- **C. CONGESTION RESULTS 49**
  - **1. Node Health Tracking 50**

2. Node Health Identification 54
a. Congestion Threshold Establishment 56
b. Node Congestion Identification at Threshold Crossing 57
D. CONGESTION ESTIMATION METHOD 60
1. Algebraic Connectivity Boundary Equations 61
2. Accuracy of Congestion Estimation 63
a. Single Phantom Node Case 63
b. Multiple Phantom Node Case 67

V. CONCLUSIONS 79
A. SIGNIFICANT RESULTS 80
B. RECOMMENDATIONS 81

APPENDIX A. MATLAB RANDOM NETWORK GENERATION AND PLOTTING 83

APPENDIX B. MATLAB ALGEBRAIC CONNECTIVITY BOUNDARIES AND LINK CONNECTIVITY CODE 85

APPENDIX C. MATLAB CONGESTION CODE 91

LIST OF REFERENCES 97

INITIAL DISTRIBUTION LIST 101

LIST OF FIGURES

Figure 1. SDN with one controller, three hosts, and one OpenFlow switch containing a secure communications channel and flow tables, after [4]. 2
Figure 2. Three layers of the SDN stack, after [5]. 6
Figure 3. OpenFlow switch containing a communications channel and n flow tables, after [8]. 7
Figure 4. Processing pipeline to determine routing of a packet in an OpenFlow switch, after [8]. 8
Figure 5. A 17 node network with ten node core network, six access network nodes, and one disconnected node, from [23]. 20
Figure 6. Network centrality graph using the three eigenvectors associated to the three smallest non-zero eigenvalues with nodes 1, 2, and 3 as least central nodes in the network, after [23]. 21
Figure 7. Nodal influence graph of node 6 at 100 percent network capacity, after [23]. 22
Figure 8. Nodal influence in the second and third eigenindices for two random nodes in a random network as the more connected (red) node's capacity used is increased from zero to one over time. 25
Figure 9. Proposed network monitoring scheme composed of six phases and the outputs required for managing flow rules. 28
Figure 10. Random network with one phantom node and all link weights equal to one. 30
Figure 11. Example algebraic connectivity upper and right boundaries for networks with integer minimum and maximum degrees. 38
Figure 12. Network topology of a random six-node modular network with two modules and an overall probability of attachment of 0.3. 42
Figure 13. Network topology showing the order of node disconnections during the node disconnections simulation. 45
Figure 14. Algebraic connectivity of 15-node networks as individual link weights are reduced from one to zero. The plots are obtained by averaging the results of 25 random networks. 48
Figure 15. Network topology of random modular network formed by the eight node adjacency matrix with two modules and an overall probability of attachment of 0.45. 50
Figure 16. (a) Network centrality of random modular eight-node network using second, third, and fourth eigenindices (b) Nodal influence plots of all nodes of the random modular eight-node network across all eigenindices, after [29]. 52
Figure 17. Nodal influence plots for all eight eigenindices of a random modular eight node network as capacity used is increased on all links of the maximum degree node (node 1) from zero to one. 53
Figure 18. Network topology of an eight node network containing one phantom node (designated as P) attached to node 7. 55

Figure 19. (a) Initial nodal influence plots of all nodes of a random modular eight-node network with two modules and an overall probability of attachment of 0.45; (b) Nodal influence plots at one second, after node 1's link weights are reduced from one to zero; (c) Nodal influence plots at two seconds, after node 3's link weights are reduced from one to zero; and (d) Nodal influence plots at three seconds, after node 4's link weights are reduced from one to zero. 58
Figure 20. Nodal influences in the second, third, and fourth eigenindices as node 1's links go to zero from zero to one second, node 3's links go to zero from one to two seconds, and node 4's links go to zero from two to three seconds. 60
Figure 21. Algebraic connectivity boundary equations and algebraic connectivity results when (a) capacity used increased on nodes of degree five from random modular graphs containing a minimum degree of three, and (b) capacity used increased on nodes of degree eight from random modular graphs containing minimum degree of three. 62
Figure 22. (a) Network topology of a random modular ten-node network with two modules and an overall probability of attachment of 0.5 (b) Network centrality graph of network using second, third, and fourth eigenindices, after [29]....64
Figure 23. Network topology of 15 node network containing a phantom node (designated by green circle) with link weight equal to 0.5 attached to node 8 and a phantom node with link weight equal to 1.5 attached to node 3. 69
Figure 24. Network topology 16-node network containing a phantom node (designated by green circles) with link weight equal to 0.5 attached to node 8, a phantom node with link weight equal to 1.5 attached to node 3, and a phantom node with link weight equal to 2.5 attached to node 2. 73

LIST OF TABLES

Table 1. Main components of a flow entry in a flow table, from [8].....8
Table 2. List of counters, after [8]. 9
Table 3. Percent difference between estimated and measured capacity used for nodes of a random ten node network containing a phantom with link weight equal to one attached to node 4. 66
Table 4. Percent difference between estimated and measured capacity used for nodes of a random ten node network when the node is attached to phantom node with link weight equal to one. 66
Table 5. Percent difference between estimated and measured capacity used for nodes of a random fifteen node network containing two phantom nodes, one with link weight equal to 0.5 attached to node 8 and one with link weight equal to 1.5 attached to node 3.....70
Table 6. Percent difference between estimated and measured capacity used for nodes of a random ten node network containing three phantom nodes, one with link weight equal to 0.5 attached to node 8, one with link weight equal to 1.5 attached to node 3, and one with link weight equal to 2.5 attached to node 2. 74
Table 7. Percent difference between estimated and measured capacity used for three low degree nodes in a random 13 node network containing a phantom node with a degree of 0.1 attached to node 8.....76

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Software defined networking is one of the latest mainstream networking designs that has gained interest for various types of communication networks. A software defined network is designed to have applications that interface with a programmable controller while the controller interfaces with network routers and switches for routing instruction. The goal of this thesis is to propose a network monitoring scheme to be implemented as an application that accesses data from the controller to monitor network states and provide information to the controller for control and management of the network.

The proposed scheme utilizes graph theory tools for the purpose of monitoring a network. The dual basis was identified as a tool that provides two measurable metrics: network centrality and nodal influence. These metrics indicate the strength and connectivity of a node relative to other nodes in the network. Changes in these metrics are directly connected to changes in network capacity, which means that they can be monitored to identify changing network conditions.

The concept of reference node is presented and used in the proposed monitoring scheme to identify nodal conditions, such as disconnections, congestion, underutilization, and attack. The concept utilizes a capacity-static phantom node, a node that exists in the adjacency matrix for graph theory analysis but not in the physical network, to establish a congestion threshold in the dual basis. The proposed monitoring scheme compares the congestion threshold to the nodal influence of a node in the dual basis in order to determine if the node is more or less connected than the phantom node; nodes that are more connected are underutilized, while nodes that are less connected are either congested or attacked.

The proposed monitoring scheme is customizable through the use of multiple phantom nodes, which produce multiple congestion thresholds. Additionally, the scheme can be tailored by changing the phantom node's link weight value, which affects the placement of the congestion threshold in the eigenindex of the dual basis; the link weight of the phantom node determines the amount of congestion allowed on a node before it is

identified as congested. An estimation method is proposed that allows the user to estimate nodal congestion at a threshold crossing in order to select the appropriate link weight value for each phantom node to achieve desired monitoring results.

MATLAB is used to validate the graph theory tools employed in the proposed monitoring scheme. First, congestion is simulated in networks that do not have a phantom node to demonstrate the controller's ability to use the dual basis to monitor changes in nodal connectivity over time. Next, congestion is simulated in networks that contain a single phantom node to demonstrate the controller's ability to identify congestion, underutilization and attack, and to validate the concept of reference node. Then, congestion is simulated in various networks with different numbers of phantom nodes to determine the effectiveness of the congestion estimation method.

The simulation results demonstrate that network centrality and nodal influence are directly tied to link capacity and, therefore, can be monitored to identify changes in node connectivity. Additionally, they demonstrate that the phantom node establishes a congestion threshold in the dual basis and nodal conditions can be identified according to the location of the node's nodal influence in the eigenindex relative to the congestion threshold. The congestion estimation method is also validated with an average difference between estimated and measure congestion of 1.5 percent, 5.6 percent, and 5.1 percent for a network with one phantom node, two phantom nodes, and three phantom nodes, respectively.

Future research should focus on validating the proposed monitoring scheme in a software defined network test bench. This requires the monitoring scheme to be developed into an application that can be implemented in the application layer of the software defined network test bench. Some concepts identified in the proposed monitoring scheme were not fully explored in this thesis. For example, the proposed scheme requires a method to poll the network for statistics, and an additional method to identify attacks was suggested. Future efforts should focus on the evaluation of these concepts and on the development of an application that acts as an intrusion detection

system for the network and an application that determines routing changes that need to be made based on network state. These are the next major step towards implementing the proposed monitoring scheme in a large, complex network.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professors Murali Tummala and John McEachen for their support, guidance and instruction, and Lieutenant Commander Tom Parker for his brainstorming assistance and counsel throughout the thesis process. I would also like to thank the professors and faculty of the Electrical and Computer Engineering Department for providing an incredible educational experience during my time at the Naval Postgraduate School.

I am extremely grateful for the love and support of my best friend, TB. He has been there for me every step of the way and provided a continuous source of strength and motivation; I could not have done any of this without him. Finally, I would like to thank my friends and family for their invaluable love and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Communication networks continue to grow in size and complexity at a rapid rate. The latest major development for communication networks is the emergence of software-defined networking (SDN) [1]. SDN has existed for the last couple decades [2], but it was not until the recent establishment of OpenFlow as the industry standard that SDN has been adopted as a mainstream network design [1].

A. BACKGROUND

A SDN is designed to have a programmable controller that provides flow level routing instruction to the network forwarding elements [3]. This design separates the control plane from the data plane, causing switches to rely on the centralized controller for control-plane functionalities [3], [4]. The controller provides these functionalities to the switches through an open interface called OpenFlow [3].

OpenFlow is the current standard for the southbound interface in a SDN [1]. It was proposed by a group of researchers in 2008 for the purpose of creating a way to run networking [4]. In 2011, the Open Networking Foundation (ONF) was formed with the mission of promoting SDN with open standards. The introduction of the OpenFlow standard is the defining accomplishment of this organization [5]. The ONF encourages equipment vendors to include OpenFlow on network switches for deployment in SDN [4].

OpenFlow exploits a common set of routing functions that exist within most network forwarding elements. It allows a controller to program flow tables in different vendor's forwarding devices for the purpose of network routing. An example SDN is displayed in Figure 1. The SDN contains a controller, and OpenFlow switch, and three hosts. The OpenFlow protocol allows the controller to communicate to the switch via a secure channel in order to update flow tables. This allows the programmable controller to act as the network's operating system [4].

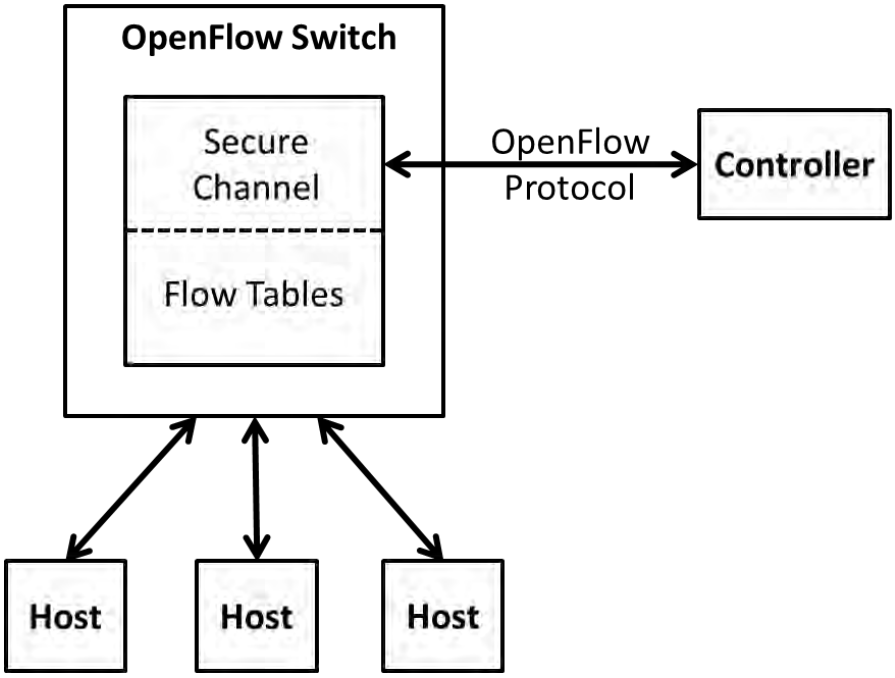


Figure 1. SDN with one controller, three hosts, and one OpenFlow switch containing a secure communications channel and flow tables, after [4].

SDNs are poised to become a dominant networking design for communication networks [6]. As shown in the network in Figure 1, OpenFlow allows complexity to be removed from network routers and switches; only the controller requires complexity since it manages the switches [3], [4]. This means large, complex networks may be implemented at a fraction of the cost [7]. It also means that robust and effective software needs to be developed and implemented on the controller to accomplish desired network tasks and functions.

B. THESIS OBJECTIVES AND APPROACH

Considering the rapid growth of interest in SDN in such areas as data centers, cloud computing, and academia [6], it is critical to have a network management system that ensures network availability. The aim of this thesis is to propose a SDN network monitoring scheme that allows a controller to provide network management functions. The scheme is designed to be implemented as an application that accesses data from the control plane of the network.

For complete and effective network management, the controller must run both monitoring functions and routing functions. The proposed scheme focuses on the controller's monitoring functions; the controller's routing decisions and routing functions are beyond the scope of this thesis because the ONF has already promoted an OpenFlow switch specification that details the process of flow rule generation for routing [8].

The main focus of the work is to identify tools or analytical techniques that are useful in modeling and analyzing network states for the purposes of monitoring a network. The proposed scheme utilizes graph theory techniques and introduces the concept of phantom nodes to produce outputs that can be used by the controller to both proactively and reactively update flow rules [8]. Both the graph theory techniques and the application of the phantom nodes are analyzed and simulated to show that the controller is able to identify various network conditions that affect network routing and management.

C. RELATED WORK

Since SDN is a new and emerging technology, a significant amount of research has been conducted on management and application with regards to OpenFlow. Management analysis for this technology was conducted in [9], [10], and [11], and different applications were discussed in [6]. While the OpenFlow interface is important to collecting statistical information on the network, an application based monitoring system that uses the northbound interface, the application programming interface (API) as described in [8] and [12], is the focus of this thesis.

Graph theory is a widely used method to model a network and has been applied to various networking problems including network robustness and connectivity optimization. Specifically, algebraic connectivity, the second smallest eigenvalue λ_2, has been proposed in [13] and [14] as a way to determine network robustness. Both [15] and [16] use algebraic connectivity to determine connectivity optimization. Additional graph theory analysis concepts, such as centrality and eigenspectrum, are explored in [17] but with no application to physical networks. These graph theory tools and analysis techniques are analyzed and expanded upon in Chapter III of this thesis.

Overall, the proposed monitoring scheme utilizes graph theory tools to monitor the dual basis for the purpose of identifying changes in nodal connectivity and congestion threshold crossings. The concept of using a phantom node to establish a congestion threshold in the dual basis has not previously been reported in the literature. In addition, this is the first known work to propose a method to estimate a node's congestion using a phantom node and known nodal degrees.

D. THESIS ORGANIZATION

The remainder of this thesis is structured as follows. The overall architecture and routing process of a SDN is described in Chapter II, along with a description of network conditions and a method to model a network using graph theory. Additional graph theory analysis tools are explored in Chapter III before presenting the proposed SDN monitoring scheme in detail. The foundational monitoring scheme concepts are simulated using MATLAB in Chapter IV, and the thesis is concluded in Chapter V with ideas and recommendations for future work. Lastly, the MATLAB code used to produce the simulation results is provided in the appendices; the code for creating and plotting random networks is provided in Appendix A, the code for connectivity simulations is provided in Appendix B, and the code for the congestion simulations is provided in Appendix C.

II. BACKGROUND

SDN is a new emerging technology [1] that has opened the doors to many areas of research [3], [9], [15]. The network management piece of that research is the focus of this thesis. In order to delve into analysis, solutions, and results, the relevant background information is presented in this chapter to introduce the reader to tools and concepts that are used and explored in later chapters. First, the basics of SDN are discussed. The architecture, including physical components and interfaces, is explained, followed by an overview of the routing process and procedures. Then, network congestion and attack are described. These are two network conditions that exist within a dynamic SDN that must be accounted for when providing network management functionalities. Lastly, graph theory is discussed because it is used in later chapters to model the SDN network. Specific focus is made on spectral graph theory because it provides analysis tools that are used in Chapter III for network monitoring.

A. SOFTWARE DEFINED NETWORKS

SDNs provide a new and innovative method to simplify network hardware by logically centralizing the routing functions of a network at the SDN controller. This means the routing algorithms and protocols are removed from the network switches and placed on the SDN controller. The controller acts as the network's operating system by providing routing and packet forwarding instructions to all the simplified switches under its control. This master-slave construct is accomplished through the design of the SDN stack and the route generation process [4], [8], [18].

1. Architecture

In order to separate the control plane from the data plane, a SDN network is organized into three layers, as shown in Figure 2. At the base, the infrastructure layer contains the physical topology of the network. This layer is the data plane and is composed of the forwarding devices or switches. The middle layer is the control layer; the control layer is the control plane of the network. The top layer is the application layer where different applications can be implemented to interface with the control plane [5].

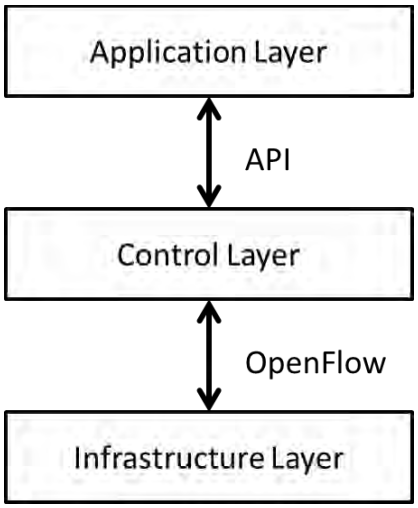


Figure 2. Three layers of the SDN stack, after [5].

The three layers of the SDN stack are able to communicate and exchange data via interfaces. The northbound interface is between the application layer and control layer; the API provides data exchange for the northbound interface. This interface has not been standardized; ONF currently has a working group exploring this interface [5], [12]. As the standardized southbound interface, OpenFlow provides data exchange between the control layer and the infrastructure layer. It allows the controller to update routing tables in the switches, which provides routing functions for the network. Switch specifications have been published by ONF to ensure the proper function of the routing process across different vendor's forwarding devices [8].

2. Routing

For the purposes of routing, the OpenFlow protocol is used to exchange information between the control plane and the data plane in order to program routing tables in the switches. The main components of an OpenFlow switch are shown in Figure 3. The switch contains multiple flow tables and a secure communications channel. The flow tables contain flow entries, which provide the switch with packet routing information. The controller is able to exchange data via the secure channel for the purpose of changing the flow entries within the flow tables. This ability provides

proactive and reactive programmable routing to the network. The switch then uses the flow tables to match flow rules for the purpose of routing incoming packets [8].

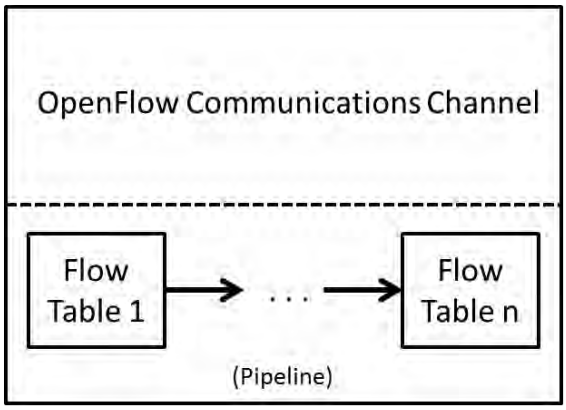


Figure 3. OpenFlow switch containing a communications channel and n flow tables, after [8].

a. Packet Matching

A switch uses the programmed flow tables to match incoming packets to flow entries, or flow rules, for the purpose of packet forwarding. If multiple flow tables exist in a switch, the flow tables are organized according to priority. A switch processes an incoming packet and attempts to match the packet to a flow rule in one of the tables in priority order. This process is known as the pipeline process [8].

The detailed process that occurs during the pipeline process is shown in Figure 4. The switch receives an incoming packet and then attempts to match that packet to a flow rule within a flow table. The tables are searched in priority order; "Table 0" has the most priority, and "Table n" has the least priority. Within the flow table, the ingress port, metadata, and packet header data are used to match a packet to a flow rule. If the switch finds a match, it performs the action associated to the flow rule. Some actions direct the switch to a different flow table for another action, but again table priority is taken into account. When the switch finds a match that has an action that does not direct the switch to a different flow table, the pipeline matching process stops and the switch then executes

that action by forwarding the packet. If a match is not found, the switch either asks the controller to provide a flow rule or drops the packet depending on switch configurations [8].

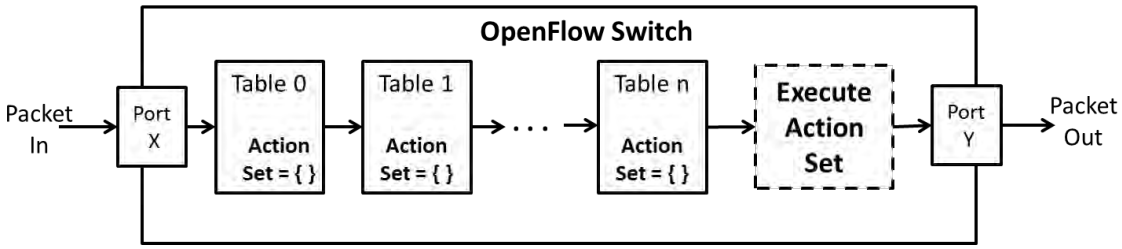


Figure 4. Processing pipeline to determine routing of a packet in an OpenFlow switch, after [8].

b. Flow Entries

The switch uses a flow entry in the pipeline process to match a packet to a specific action set; however, the flow entries contain additional information set by the controller and used by the switch to manage the entries. The different fields contained within a flow entry of a flow table are displayed in Table 1. The *match* field contains the entry port, packet headers, and optional metadata and is used by the switch to determine an action set. The *priority* field is used to match flow entry precedence. The *counter* field is used to update specified counters. The counters can be used by the controller to determine the amount of traffic moving across the switches. The controller uses the *instructions* field to change action sets or modify pipeline processing. The *timeout* field contains the time to live (TTL) information for the specific flow entry. If the TTL expires, the flow entry is removed from the flow table. Lastly, the *cookie* field is used by the controller for filtering and management processes. Each of these fields provides a unique set of data that aids in the flow rule generation process [8].

Table 1. Main components of a flow entry in a flow table, from [8].

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

c. Polling the Switch

For routing management, the controller requires information from the switch to better understand traffic loads throughout the network. The read-state message provides the controller with the ability to collect the statistical information from the switch that is stored within the counters field of the flow tables. The controller sends the read-state message via the secure channel using OpenFlow protocols, and the switch is required to respond with information regarding statistics, capabilities, and configurations [8]. The required and optional counters that are found on OpenFlow switches are displayed in Table 2. The information stored in the counters field is sent to the controller during statistical polling. Since the counters are configurable by the controller [8], the required counters can be set to collect information needed for determining current network states.

Table 2. List of counters, after [8].

Counter	Bits	
Per Flow Table		
Reference count (active entries)	32	Required
Packet Lookups	64	Optional
Packet Matches	64	Optional
Per Flow Entry		
Received Packets	64	Optional
Received Bytes	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Port		
Received Packets	64	Required
Transmitted Packets	64	Required
Received Bytes	64	Optional
Transmitted Bytes	64	Optional
Receive Drops	64	Optional
Transmit Drops	64	Optional
Receive Errors	64	Optional
Transmit Errors	64	Optional
Receive Frame Alignment Errors	64	Optional
Receive Overrun Errors	64	Optional
Receive CRC Errors	64	Optional
Collisions	64	Optional
Duration (seconds)	32	Required

Duration (nanoseconds)	32	Optional
Per Queue		
Transmit Packets	64	Required
Transmit Bytes	64	Optional
Transmit Overrun Errors	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Group		
Reference Count (flow entries)	32	Optional
Packet Count	64	Optional
Byte Count	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Group Bucket		
Packet Count	64	Optional
Byte Count	64	Optional
Per Meter		
Flow Count	32	Optional
Input Packet Count	64	Optional
Input Byte Count	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Meter Band		
In Band Packet Count	64	Optional
In Band Byte Count	64	Optional

Overall, the architecture and routing process allow the SDN to separate the data plane from the controller plane [5], [18]. This means network costs can be decreased by using inexpensive switches in conjunction with a programmable controller. The programmable controller, however, must then be capable of identifying and handling different network conditions, specifically congestion and attack, in order to update flow rules in the flow tables for the purposes of network management. These two network conditions are discussed in the next section.

B. NETWORK CONGESTION AND ATTACKS

Network congestion and network attack are the two main conditions that must be monitored in a SDN to provide effective network management. Congestion occurs when a node is overwhelmed by the number of packets that must be processed. Each node and

network has a packet-handling capacity and, in general, congestion is defined by a node or network that reaches 80 percent of its packet-handling capacity [19]. When a packet arrives at a switch, it is examined for match data. That data is then used to match a flow rule and determine an action. Based on the action, the packet is then sent to the appropriate output buffer [8]. When multiple packets arrive at a switch, the packets are queued in memory until they can be processed. If the rate of packet arrival exceeds the rate of packet processing and packet transmission, the queue size grows and delay is experienced. At approximately 80 percent queue utilization, the queue length and delay begin to grow at extreme rates. If a congestion control mechanism is not implemented, the queue eventually overflows [19] and the node fails because it cannot keep up with network traffic.

An attack is different from congestion because it is malicious in nature, but the result of a denial-of-service (DoS) attack is congestion. In a DoS attack, the goal is to prevent or deny access to a specific service or node. For example, one type of DoS attack is a "flood" attack. The attacker overloads the node with packets until the node's queue becomes overwhelmed and the legitimate packets either experience large delays or are dropped due to a lack of space in the memory [20].

Congestion and attack are two network conditions that require action by the controller within a SDN. Both reduce or eliminate communication paths between nodes and, therefore, the controller must proactively and reactively update flow rules to combat these network conditions. A method to identify these two network conditions using graph theory analysis techniques is presented in Chapter III. The basic mathematical tools that are used in graph theory analysis are discussed in the next section.

C. GRAPH THEORY

Graph theory provides a mathematical way to represent a network. To model a network's topology, the network is defined as a graph, the hardware device is defined as a node, and the communication path between the devices is defined as a link [21]. These terms are used interchangeably throughout this thesis.

For a graph G there exists a set of nodes N and a set of links L. All links are assigned weights W [21], [22] to represent a measurable quantity like utilization U or capacity available C_A [23]. The capacity available C_A is defined as the fraction of the channel capacity that is available at any given time and is normalized to be between zero and one. Capacity used C_U is defined as the fraction of the channel capacity that is used at any given time [19], [24]:

$$C_U = 1 - C_A. \tag{1}$$

The link weight w_{ij} between the i^{th} and j^{th} node is defined as

$$w_{ij} = C_A = 1 - u_{ij} \tag{2}$$

where u_{ij} is the link utilization between node i and node j, defined as

$$u_{ij} = \frac{c_m(t)}{c_{\max}} \tag{3}$$

where $c_m(t)$ is the measured channel capacity as a function of time, and $c_{\max}$ is the fixed maximum channel capacity for that link [19], [23], [24].

There are two types of graphs: complete graphs and incomplete graphs. In a complete graph K_n, a link exists between every set of nodes [22]. If a link does not exist between every set of nodes, the graph is considered incomplete. Each network has two connectivity metrics: node connectivity k_N and link connectivity k_L. Node connectivity k_N is defined as the minimum number of nodes and adjacent links that have to be removed to disconnect the graph. Link connectivity k_L is the minimum number of links that have to be removed to disconnect the graph [13].

To model a network as a graph, the topological information is used to create the adjacency matrix A, the degree matrix D, and the Laplacian matrix Q. The $n \times n$ adjacency matrix contains n nodes and is used to describe the network topology. Each element a_{ij} within A is determined by

$$A(i,j) = \begin{cases} w_{ij} & \text{if } a_{ij} \in L \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where w_{ij} is the link weight between the i^{th} and j^{th} node and L is the set of all links in the network. The $n \times n$ degree matrix is used to determine the degree d_{ij} or number of links, for each node, and is defined as

$$D(i,j) = \begin{cases} \sum_{j=1}^{n} w_{ij} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The $n \times n$ Laplacian matrix is then defined as

$$Q = D - A \tag{6}$$

or [21], [25]

$$Q(i,j) = \begin{cases} -w_{ij} & \text{if } a_{ij} \in L \\ \sum_{j=1}^{n} w_{ij} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

It is important to note that the adjacency matrix, degree matrix, and Laplacian matrix represent undirected graphs. This means direction is not included or indicated by a link; a link simply specifies that a connection or communication path exists between two nodes. Directed graphs can be represented using the incidence matrix [21], but this concept is not explored in this thesis. For the purposes of this thesis, flow modeling is not necessary for spectral analysis.

1. Spectral Graph Theory

Spectral graph theory, a field within graph theory, uses graph characteristics to provide spectral analysis. The eigenvalues and eigenvectors of the adjacency and Laplacian matrices are examined to derive graph characteristics [21], [26], [27]. The eigenvalues and eigenvectors of the $n \times n$ Laplacian matrix Q are defined as the solution to

$$Qv_i = \lambda_i v_i, \quad i = 1, 2, ..., n \tag{8}$$

where λ_i is the i^{th} eigenvalue, v_i the corresponding $n \times 1$ eigenvector, and i is the eigenindex [21].

The order of the eigenvalues is

$$0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{n-1} \leq \lambda_n. \tag{9}$$

Due to the positive, semidefinite characteristic of the Laplacian matrix,

$$\lambda_1 = 0 \tag{10}$$

and

$$v_1 = 0 \tag{11}$$

for all graphs [21]. Additional zero eigenvalues indicate node disconnections; the total number of disconnected nodes N_d is defined as

$$N_d = 1 - I_z \tag{12}$$

where I_z is the largest eigenindex i for which $\lambda_i = 0$ [14], [21]. The set of zero eigenvalues $\{\lambda_i = 0\}$ define the null space, which represents the part of the graph that cannot be reached [21], [23]; the set of non-zero eigenvalues $\{\lambda_i \neq 0\}$ define the reachability space, which represents the part of the graph that has a guaranteed communication path. Additionally, the rank of the null space is N_d and the rank of the reachability space is $n - N_d$ [23], [28].

The n elements of v_i are represented as v_i^l, which is the l^{th} element of the i^{th} eigenvector, and the n eigenvalue-eigenvector solutions can be represented in matrix form as

$$Q = V \Lambda V^T \tag{13}$$

where Λ is the $n \times n$ diagonal matrix of the eigenvalues and V is the $n \times n$ orthonormal matrix formed by concatenating the n eigenvectors [23], [28]. The eigenspace described by Λ and V provides graph characteristics that can be used to determine such information as robustness [13], [14], connectivity optimization [15], [16], and centrality [17].

Overall, graph theory and spectral graph theory provide tools to model and analyze networks for the purposes of determining states or conditions. Each row of the adjacency matrix A is directly tied to a specific node [21], and network conditions

experienced by that node are accounted for in the link weights of A [23]. Both the degree matrix D and the Laplacian matrix Q are calculated from A, which means that the rows in each of these matrices are directly tied to a specific node as well. As a result, the eigenvalue matrix Λ and eigenvector matrix V are also affected by changes in link weight values [21], [28]. To sum up, graph theory provides tools that can be exploited for network analysis, control and management of a network.

The background information related to SDN, network conditions, and ways to model networks mathematically was covered in this chapter. SDNs are designed to have a logically centralized operating system that controls and manages network conditions by setting flow rules within flow tables. The centralized controller, therefore, must have a method to analyze and identify network conditions. Graph theory is identified and discussed in the next chapter as a tool for providing network management capabilities to a SDN controller. A graph theory based monitoring scheme that provides outputs to allow the controller to self-manage the network is described in Chapter III.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROPOSED NETWORK MONITORING SCHEME

Within the SDN framework, the centralized controller is responsible for determining and updating flow rules that dictate how packets are routed in the network. The controller collects statistical data on the current state of the network to develop and update these flow rules. This process allows the controller to provide adaptive traffic flow management for the entire network [4], [8], [27]. In order for this process to work, the controller must be able to monitor the network's state with the goal of identifying, locating, and resolving congestion. Current network analysis and management concepts are either incompatible with the SDN framework or not computationally efficient [27]. A new monitoring scheme that uses a computationally efficient method to monitor network states and provide congestion management for a SDN is proposed in this chapter.

The remainder of this chapter is organized as follows. The dual basis is examined first, including the concepts of the eigencentrality basis and the nodal basis. These provide the foundation for the tools and concepts used both the reference node concept and network analysis in the proposed monitoring scheme. Next, phantom nodes are defined and the reference node concept is presented. Phantom nodes provide the controller with a tool to measure and determine network conditions. Then, the proposed monitoring scheme is presented. The scheme utilizes phantom nodes and eigenanalysis to determine network connectivity and congestion. The chapter concludes with an examination of a method to estimate congestion at a phantom threshold crossing in the proposed monitoring scheme.

A. DUAL BASIS

Eigenvalues and eigenvectors reveal a wealth of information about network states and node interactions that can be used to monitor and manage a network. The dual basis provides two perspectives on the link between eigenvalues and eigenvectors [17], [23]. With these links, a controller can gain detailed knowledge of network connectivity and congestion in order to update flow rules.

1. Orthogonality

Effective network management stems from the controller's ability to detect and combat adverse network conditions through identification of changes in nodal states. The dual basis allows a controller to perform a detailed analysis of these states at a given instant and over time. One characteristic of the Laplacian matrix is that it always forms a dual basis [28]. The dual basis produces two orthogonal bases that provide information on how much influence an individual node has on a particular eigenvalue and what influence an individual node has across the entire range of eigenvalues [23]. By monitoring the changes in both bases over time, we can determine a node's connectivity and determine and track centrality continuously.

Given an orthonormal matrix of eigenvectors V , we have

$$V^T V = I \tag{14}$$

or

$$V^{-1} = V^T \tag{15}$$

where I is the identity matrix [28]. The right eigenvector representation

$$QV = V\Lambda \tag{16}$$

is obtained by multiplying both sides of (13) on the right with V . Conversely, the left eigenvector representation

$$V^T Q = \Lambda V^T \tag{17}$$

is obtained by multiplying both sides of (13) on the left by V^T [23]. The right eigenvector transformation and left eigenvector transformation produce two orthogonal bases, which can be defined as

$$QV = Z_1 \tag{18}$$

$$V^T Q = Z_2 \tag{19}$$

where Z_1 is termed the eigencentrality space and Z_2 the nodal space [17], [23]. The eigencentrality basis describes the amount of influence each node has on a single eigenvalue. The nodal basis, on the other hand, describes a single node's influence across all eigenvalues [23]. Together, these two spaces define two distinct perspectives of

network robustness, centrality and connectivity and form the foundation for using eigenanalysis in the proposed monitoring scheme.

2. Eigencentrality Basis and Nodal Basis

The eigencentrality basis and nodal basis form a holistic representation of the state of all nodes in the network. The eigencentrality basis characterizes network centrality, while the nodal basis characterizes nodal influence. Both of these characteristics provide connectivity metrics that measure a node's health and strength within a network [17], [23].

a. Network Centrality

Network centrality uses the eigencentrality basis to measure a node's importance within a network by determining how central it is for network connectivity. A node that is more central has a larger impact on network connectivity if it is removed or disconnected than a node that is less central in the network [23]. Network centrality is determined by the $n \times n$ Laplacian matrix Q and the n eigenvector solutions from (8) that form the $n \times n$ eigenvector matrix V [29].

Network centrality is graphically represented as a 3- or 2-dimensional plot. A three dimensional representation is obtained by plotting (v_i^l, v_j^l, v_k^l), for $l = 1, 2, ..., n$, where i, j, and k are three consecutive eigenindices [29]. Each (v_i^l, v_j^l, v_k^l) coordinate depicts the l^{th} node's centrality; v_i^l is tied to the l^{th} row of the adjacency matrix A, which in turn is associated to a specific node [17], [21], [23], [28]. The Laplacian matrix Q is used to produce the links between the nodes [29].

The smallest eigenindices produce a graph with the most central nodes in the middle and the least central nodes at the edges [23], [30]. This separation provides a metric to determine nodes that are more likely to be adversely affected by network conditions, such as congestion. Nodes that are plotted in the middle of the graph have more links to combat congestion than the nodes plotted at the edges. The larger

eigenindices can also be used, but they produce graphs that have the most central nodes in the middle and the least central nodes at the edges [23], [30].

Consider the network shown in Figure 5 containing ten blue nodes representing a core network, six green nodes representing access networks, and one red node representing a disconnected network. An example network centrality graph for this network is shown in Figure 6. Since node 17 is disconnected and two zero eigenvalues exist, the network centrality is graphed using the three 17×1 eigenvectors associated to the three smallest nonzero eigenvalue solutions (v_3^l, v_4^l, v_5^l) [23], [29].

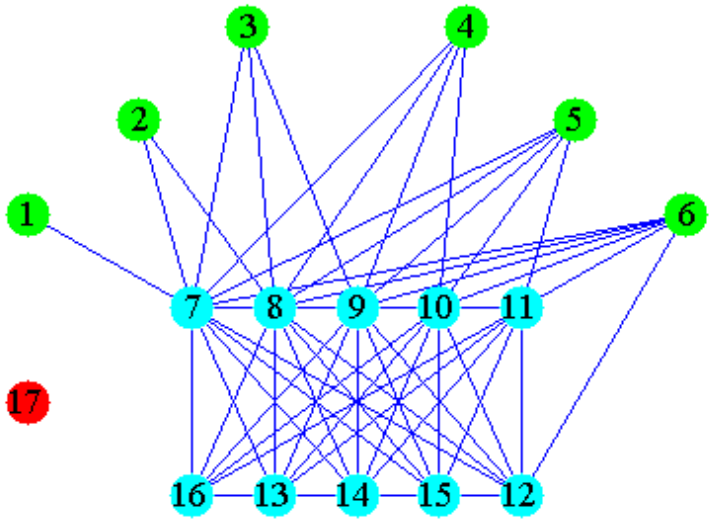


Figure 5. A 17 node network with ten node core network, six access network nodes, and one disconnected node, from [23].

As shown in Figure 6, the graphed results provide a good visual representation of the connectedness as described by the eigencentrality basis and network centrality. Nodes 1, 2 and 3 are the least connected nodes in the network and are plotted away from the middle of the graph. The remaining nodes are more connected and are plotted in the middle of the graph [23], [30]. The disconnected node is plotted at $(0,0,0)$ because it is in the null space [21], [23]. Overall, the graphical representation of network centrality

shown in Figure 6 illustrates that the eigencentrality basis provides information on the connectedness, strength, and centrality of the nodes in the network.

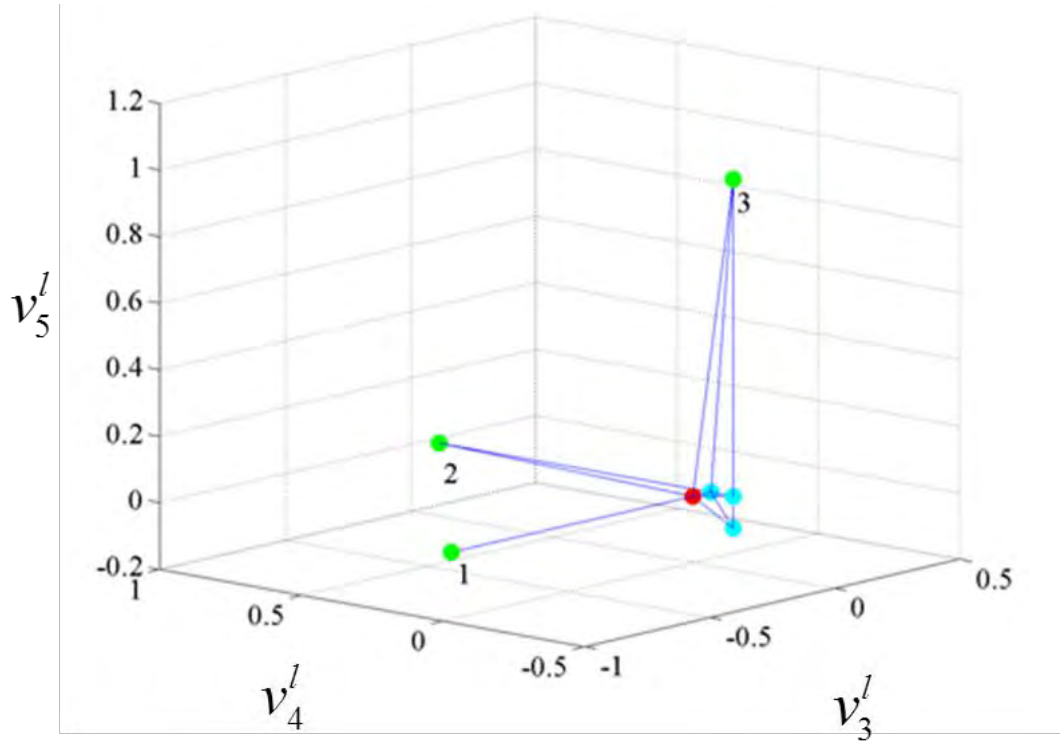


Figure 6. Network centrality graph using the three eigenvectors associated to the three smallest non-zero eigenvalues with nodes 1, 2, and 3 as least central nodes in the network, after [23].

b. Nodal Influence

Nodal influence is a measure of a node's impact on a specific eigenvalue solution λ_i and is quantified as $(v_i^l)^2$. It is determined by the eigenvector elements that form the nodal basis. From (13), the l^{th} diagonal entry of the Laplacian matrix Q can be shown as

$$Q(i,i) = (v_1^l)^2\lambda_1 + (v_2^l)^2\lambda_2 + (v_3^l)^2\lambda_3 + \ldots + (v_n^l)^2\lambda_n \tag{20}$$

where v_1^l is the l^{th} element of the i^{th} eigenvector, and $Q(i,i)$ is the degree of the i^{th} node [21], [28]. This means that a node's degree or connectivity directly determines the nodal influence that the node has on an individual eigenvalue λ_i.

The nodal basis is graphically represented by plotting the set of coordinates $(i,(v_i^l)^2)$ for a specific l. Each l produces a different nodal influence graph because the $(v_i^l)^2$ values are dependent on the degree and connectedness of the node associated to the l^{th} element of the eigenvector matrix V [17], [21], [28]. An example nodal influence graph is shown in Figure 7 for node 6 from the network in Figure 5. Each plotted stem is the $(v_i^6)^2$ value for that particular eigenindex i. The graph shows that node 6 has a large nodal influence on the eighth eigenvalue and a small nodal influence on the remaining eigenvalues.

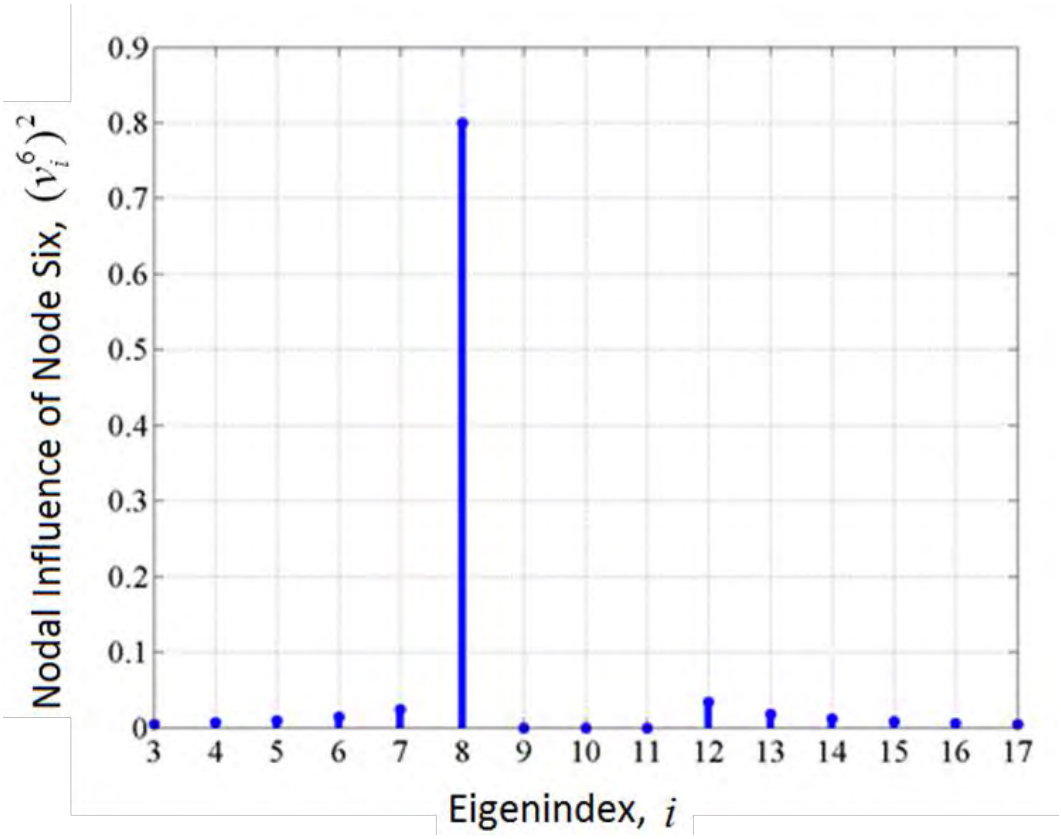


Figure 7. Nodal influence graph of node 6 at 100 percent network capacity, after [23].

Since the eigenvalues are ordered according to (9) and the value of an eigenvalue represents connectedness, larger eigenvalues are associated to more connectedness and smaller eigenvalues are associated to less connectedness [17], [21]. Relating this concept

to (20), we see that nodes that have larger nodal influence values for the larger eigenvalues are more connected than the nodes that have larger nodal influence values for the smaller eigenvalues. This perspective decouples nodes in the network to allow for analysis of the strength and connectivity of individual nodes [23]; the changes in an individual node's influence can be monitored over time to determine changes in connectivity.

The eigencentrality basis and the nodal basis, together, provide the foundational tools for network analysis. They show that connectivity and node health can be determined from the eigenvectors and eigenvalues [23]; therefore, a controller can gain insight into a network's state if it is programmed to track and analyze the changes in the eigenvalue and eigenvector matrices. This basic concept is utilized in conjunction with phantom nodes to provide the controller with a method of identifying adverse network conditions.

B. PHANTOM NODES

Most existing applications of graph theory focus on networks in a static state [13]–[16]. The idea of comparing changes in the eigenspectrum over time to determine changes in nodal connectivity has not been reported in literature previously. To implement this in a dynamic network, we introduce a new concept that utilizes phantom nodes. The use of phantom nodes is crucial to the proposed monitoring scheme discussed later in this chapter.

A phantom node is defined as a node that appears within the adjacency matrix but does not physically exist within the network topology. The phantom node is a capacity-static node with a link weight that is set by the user during network initialization. The link weight causes the phantom node to have the largest nodal influence of a particular eigenvalue within the eigenindex [21]; the largest nodal influence within an eigenvector is defined as $v_i^{l\max}$ where the $l_{\max}{}^{th}$ element is the largest squared value in the i^{th}

eigenvector. The phantom node's $v_i^{l\max}$ establishes a threshold in the eigenindex that can be used to compare nodal connectivity; this is the foundation for the reference node concept.

1. Reference Node

The concept of reference node is to identify nodal states by comparing the nodal influence of a node to that of the phantom node; the identification of nodal conditions is discussed later in this chapter. Since the location of $v_i^{l\max}$ in the eigenindex i indicates a node's connectivity relative to other nodes in the network [17], [21], [23], the establishment of the phantom node with $v_i^{l\max}$ for a specific eigenindex allows the connectivity of other nodes to be compared to the phantom node; the phantom node's connectivity never changes since its capacity remains static.

Applying the concept of reference node, we see that a node that has $v_i^{l\max}$ in a smaller eigenindex compared to the phantom node is less connected than the phantom node. Conversely, a node that has $v_i^{l\max}$ in a larger eigenindex compared to the phantom node is more connected [17], [21]. In a dynamic setting, the nodal influence is monitored to determine if a node is more or less connected than the phantom node. For example, if the phantom node is initially established to have $v_2^{l\max}$ and congestion forces a node to become weaker than the phantom node, then that node takes over primary influence of λ_2 and has $v_2^{l\max}$. The phantom node becomes the primary influencer of λ_3, which means that it now has $v_3^{l\max}$. If an additional node becomes weaker than the phantom node, another shift in eigenvalue influence occurs. Conversely, the opposite shift occurs if the node grows stronger or becomes less congested.

An example of a shift in nodal influence for a random dynamic network is illustrated in Figure 8. The values of $(v_2^l)^2$ and $(v_3^l)^2$ are plotted against the node's capacity used C_U for two nodes in a random network. As capacity available C_A is decreased on the node associated to the red line, its degree and connectivity decrease,

resulting in a decrease in $(v_3^l)^2$ and an increase in $(v_2^l)^2$. At the same time, the $(v_3^l)^2$ increases and $(v_2^l)^2$ decreases for the blue node because the relative difference in connectivity between the red and blue nodes decreases. At $C_U = 0.8$, a shift in nodal influence occurs in the second and third eigenindices; the red node takes over nodal influence in the second eigenindex $i = 2$ and the blue node takes over nodal influence in the third eigenindex $i = 3$. A shift in the nodal influence $(v_i^l)^2$ within the eigenindex i indicates that a node has become weaker or stronger, depending on the direction of the changes, than another node in the network.

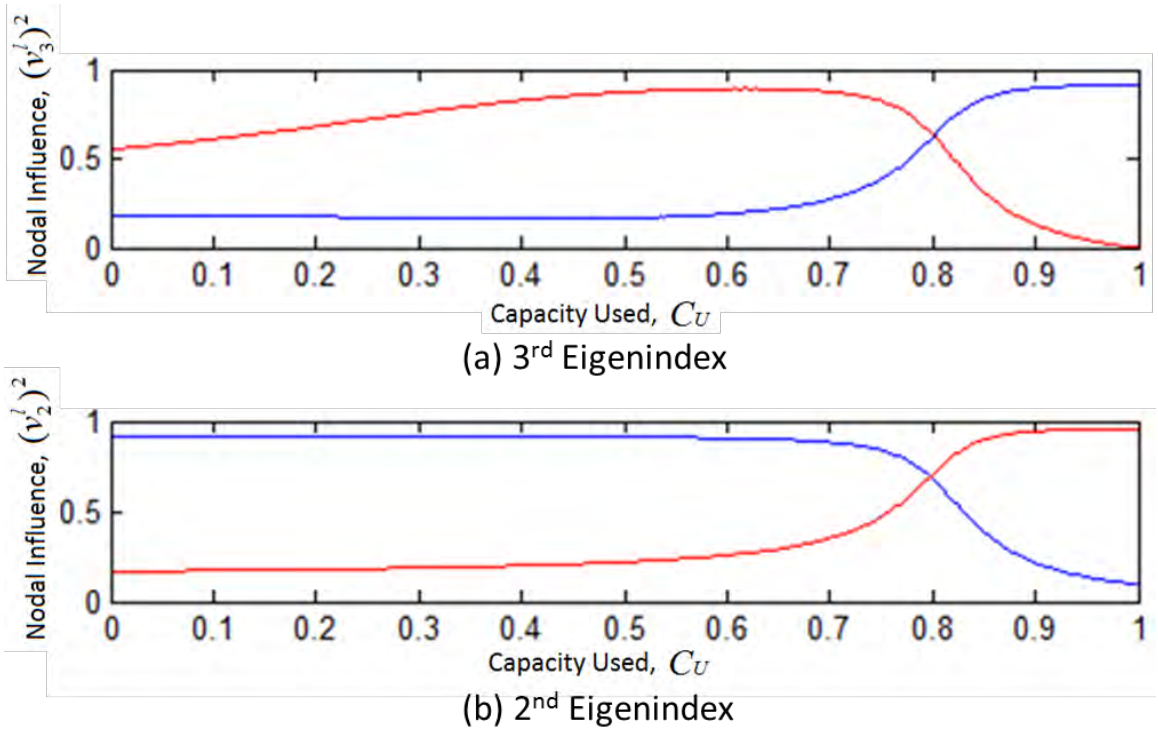


Figure 8. Nodal influence in the second and third eigenindices for two random nodes in a random network as the more connected (red) node's capacity used is increased from zero to one over time.

A shift in nodal influence within the eigenindex i between a node and the phantom node is defined as a congestion threshold crossing. In the proposed scheme, the controller identifies congestion threshold crossings by tracking the changes in the nodal

influence and determines a node's state based on the nodal influence of the node relative to the congestion threshold. This concept is discussed in more detail later in this chapter.

It is important to note that it is possible for a node to have primary nodal influence over multiple eigenindices i; primary nodal influence over multiple eigenindices typically occurs for the middle eigenindex values when a node is neither strongly connected nor weakly connected. This does not affect the use of the congestion threshold because nodes still have either primary nodal influence for a larger or smaller eigenindex than the group of eigenindices for which the phantom node has primary influence.

2. Placing the Congestion Threshold

The phantom node can be placed to exercise control over any eigenvalue by selecting the appropriate link weight; the link weight value determines the congestion threshold level, which in turn determines the amount of allowed congestion for each node in the network. The method for estimating allowed congestion for a given threshold is discussed later in this chapter.

In order to be used as a threshold for all nodes in the network, the phantom node must have $v_2^{l\max}$; the phantom node is the least connected node in the network which means that it acts as a connectivity threshold for all nodes in the network. If the phantom node was positioned to have $v_3^{l\max}$, one node will always be identified as less connected than the phantom node; the less connected node will not be identified as "crossing" the established threshold, which is a key concept used in the proposed monitoring scheme to identify congested, underutilized, and attacked nodes.

Multiple phantom nodes can be used to produce additional congestion thresholds in the eigenindex. Multiple thresholds are proposed for networks that have a large difference between the minimum degree $D_{\min}$ and maximum degree $D_{\max}$ of the nodes. In these cases, a small degree node may be identified as congested at a reasonable congestion level, while a larger degree node may not be identified as congested until it is

near failure. Including an additional phantom node solves this problem by producing additional thresholds that are used for a subset of the network's nodes. This is illustrated in Chapter IV.

It is also important to note that a link weight value must be selected that initially gives the phantom node primary nodal influence over a single eigenindex i, not shared primary influence over multiple eigenindices. Shared nodal influence occurs when eigenvalues are repeated or when the connectivity of two nodes is similar; therefore, the phantom node must have an initial link weight value that is not equal to the degree of any nodes in the network to ensure the placement of the threshold. In most cases, this means that the phantom node is assigned a non-integer link weight.

Overall, a phantom node provides a measurable connectivity metric that is used in the proposed monitoring scheme to identify network conditions. The proposed monitoring scheme which utilizes the dual basis, the concept of reference node, and the congestion threshold crossing to identify network disconnections, congested nodes, underutilized nodes, and attacked nodes is presented in the next section.

C. PROPOSED MONITORING SCHEME

For effective network management, the SDN controller must use a method that monitors the network's state, analyzes connectivity and congestion, and takes action to resolve congestion and connectivity problems. A SDN monitoring scheme that allows a controller to manage a network by identifying and locating congestion, network connectivity issues, link failures, and nodal failures due to attack or hardware malfunction is proposed in this section.

The proposed phantom-based monitoring scheme is shown in Figure 9. The scheme's loop is set in motion upon completion of network initialization. During network initialization, the network administrator sets up the network and defines the location of the threshold. Once in motion, the scheme cycles through the six phases producing outputs, which are used to determine routing decisions.

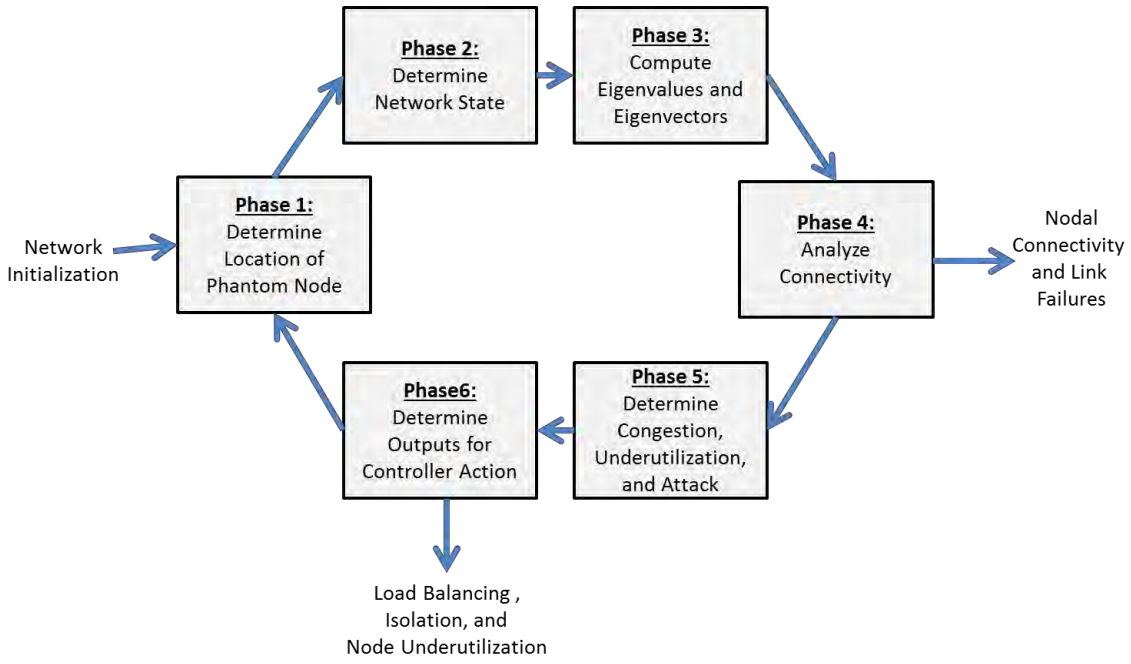


Figure 9. Proposed network monitoring scheme composed of six phases and the outputs required for managing flow rules.

1. Determine Location of Phantom Node

The physical placement of the phantom node in the network is determined by the node that has $v_n^{l\max}$, which is the most connected node within the dual basis because it has primary nodal influence of the largest eigenvalue [17], [21], [23]. It is also the node furthest from the congestion threshold in the nodal basis [23] and the least likely to be overcome by congestion. At network initialization, the controller does not have the eigenvectors calculated, so the phantom node is attached to the node with the maximum degree $D_{\max}$; the node with $D_{\max}$ has $v_n^{l\max}$ in a network at 100 percent capacity available C_A [17], [21]. Otherwise, the phantom node attaches to the node that has the largest nodal influence on the largest eigenvalue other than the node previously attached to the phantom node.

The physical placement of the phantom node in the network is critical for the proposed monitoring scheme because the degree of the node attached to the phantom node is altered, which in turn affects the number and order of the eigenvalues and

eigenvectors and the values of the elements within the eigenvalue and eigenvector matrices [16]. This change in degree causes the node to appear stronger and more connected in the dual basis. Since the nodal basis is altered, the controller cannot accurately detect adverse network conditions for the node attached to the phantom node using the reference node concept. To solve this problem, the location of the phantom node needs to continually change within the network to ensure node congestion, overutilization, or failure do not go undetected by the SDN controller. Once the phantom node is placed or relocated, the controller can begin determining network state.

2. Determine Network State

The network state is determined by the updated adjacency matrix A with the current C_A for all links; all graph theory matrices used for network analysis are calculated from A [21]. Current link weights are determined based upon (2) and (3) [19], [23]. This means that the controller requires a mechanism to measure the channel capacity for each individual link to determine $c_m(t)$. The mechanism for measuring link capacities is outside the scope of this thesis, but several link measurement techniques, including a passive measurement technique, have been proposed in literature for SDN [3].

It is also important to note that connectivity analysis in Phase 4 of the proposed monitoring loop requires a minimum capacity available $C_{A\,\min}$ threshold to be implemented in the adjacency matrix A for link connectivity analysis. Since $C_{A\,\min}$ is defined as the minimum capacity required on a link for communications to exist, $C_{A\,\min}$ determines whether a node should be considered connected; if $w_{ij} \leq C_{A\,\min}$, where w_{ij} is the link weight determined by (2), then the controller sets that link to zero in the adjacency matrix A.

Consider the network shown in Figure 10 containing five blue nodes and one green phantom node. This network is used to demonstrate the purpose and value of the $C_{A\,\min}$ threshold. If all of node 2's link weights drop to 0.0001, this node does not have enough C_A to communicate on any of the links [19] and, therefore, has become

disconnected from the network; however, the eigenvalues are $\{0, 0.004, 0.6973, 1.3820, 3.6181, 4.3028\}$, which means the network is connected because a second zero eigenvalue does not exist [14], [21].

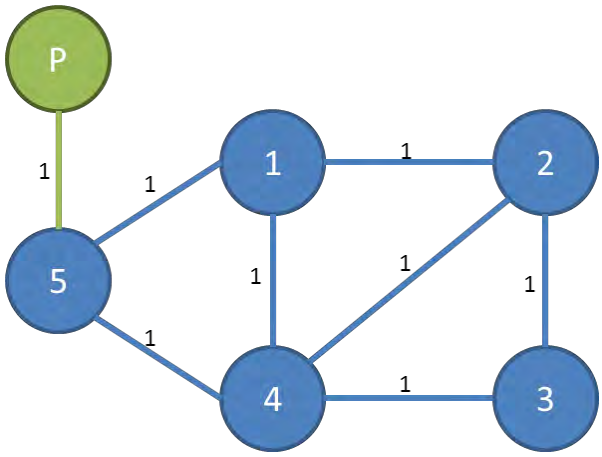


Figure 10. Random network with one phantom node and all link weights equal to one.

3. Compute Eigenvectors and Eigenvalues

The eigenvalue matrix Λ and eigenvector matrix V are calculated from the adjacency matrix A using (6) and (8) [21], [28]. This calculation is more difficult for larger networks but is not outside the capabilities of a controller that has a modern processor. Depending on the algorithm employed for this calculation, the values of the elements of an eigenvector in the null space may vary; given a zero eigenvalue, (8) becomes

$$Qv = 0 \tag{21}$$

and numerous solutions exist that satisfy (21) [28]. Additionally, the elements of V produced from (8) contain negative magnitudes [21], [31]; V contains all eigenvector elements v_i^l [21] and $(v_i^l)^2$ is needed to determine nodal influence and be used in the reference node concept [17], [21], [23]. Therefore, a standard eigenvector matrix V_s is introduced for the proposed monitoring scheme to make nodal analysis easier.

To format V into V_s, negative magnitudes and zero eigenvalues must be addressed. Negative magnitudes occur because the values in each eigenvector sum to zero [17], [21], [31]. Following principle component analysis, we must square each value in the eigenvector [17]. This causes all v_i^l to become $(v_i^l)^2$, which removes the negative magnitudes. This is important for connectivity and congestion analysis using the dual basis.

To address the numerous solutions produced for a zero eigenvalue case, the null space and reachability space of the eigenvector matrix V are examined. If $v_i^l = 0$ for a particular l across all eigenindices i in the reachability space, then the l^{th} elements of the null space are investigated for the element with a magnitude closest to one. The element closest to one is set to one, and the remaining elements in that vector are set to zero. Additionally, the remaining l^{th} elements in the null space are set to zero. Once this process is complete, one eigenvector remains unformatted in the null space and is associated to the zero eigenvalue that always exists at λ_1. All elements in this eigenvector are set to zero.

The network shown in Figure 10 is used to illustrate the standard eigenvector matrix V_s. If node 2 links are dropped, an additional zero eigenvalue occurs. The resulting eigenvalues are $\{0, 0, 0.6972, 1.3820, 3.6180, 4.3028\}$, and the resulting V_s is

$$V_s = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.6325 & 0.6325 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.6768 & 0.5117 & 0.1954 & 0.2049 \\ 0.0000 & 0.0000 & 0.2049 & 0.1954 & 0.5117 & 0.6768 \\ 0.0000 & 0.0000 & 0.2049 & 0.1954 & 0.5117 & 0.6768 \\ 0.0000 & 0.0000 & 0.6768 & 0.5117 & 0.1954 & 0.2049 \end{bmatrix}. \quad (22)$$

The second eigenvector of (22) has zero entries for all v_2^l except v_2^2, which is associated to node 2. Additionally, $v_i^2 = 0$ for $i \neq 2$, which was previously set to one in the null space. The V_s provides easy identification of the nodes in the null space, which is necessary for connectivity analysis.

4. Analyze Connectivity

Connectivity analysis uses the null space to identify disconnected nodes that need to be addressed by the routing function of the SDN controller. Traditionally, algebraic connectivity has been the metric used to determine network connectivity [15], but it only identifies that a network is either connected or disconnected [31] and it does not provide information regarding the number of disconnected components or the location of the disconnected components [14], [21].

The null space is deemed to be a more effective metric for connectivity analysis. The number of disconnected nodes N_d is based upon (13), [14], [21]. The identity of the disconnected node is identified by the null space of the standard eigenvector matrix V_s because the null space contains the eigenvectors v_i linked to the zero eigenvalues $\lambda_i = 0$. If $v_i^l = 1$ in the null space of V_s, then the l^{th} node is responsible for the $\lambda_i = 0$ and is disconnected from the network; the l^{th} element of the standard eigenvector matrix V_s is associated to the l^{th} node of the adjacency matrix A [17], [21], [23], [28].

It is important to note that neither the null space nor algebraic connectivity provide information on link failures; link failures are not directly tied to algebraic connectivity [13], and the null space describes nodal connectivity [23]. Link failures are determined by elements a_{ij} of the adjacency matrix A since the minimum capacity available $C_{A\min}$ threshold sets $a_{ij} = 0$ if the link does not have enough C_A to communicate.

Connectivity analysis provides node disconnection and link disconnection information as outputs to the routing function of the controller. The route management and generation part of the controller can use this information to make changes to flow rules in order to fix connectivity problems that exist in the network.

5. Determine Congestion, Underutilization, and Attack

Nodal congestion, underutilization and attack are determined by the location of primary nodal influence $v_i^{l\max}$ in the eigenindex i. Congested and attacked nodes have

$v_i^{l\max}$ for a smaller i than the phantom node, while underutilized nodes have $v_i^{l\max}$ for much larger i than the phantom node. These determinations are made by setting the phantom node as the reference node and using the concept of a reference node as described earlier in this chapter.

Distinguishing congestion from an attack requires monitoring of changes in nodal influence in the dual basis over a period of time. Congestion decreases and capacity available C_A on a node's links increases if flow rules are implemented to decrease the amount of traffic on the links [19]. As the node experiences less traffic on its links, it becomes stronger and more connected, and the node's $v_i^{l\max}$ shifts right in the eigenindex i to influence larger values of i [23]. Eventually, the node will has $v_i^{l\max}$ for an eigenindex larger than the phantom node's $v_i^{l\max}$ if links experience enough available capacity C_A.

On the other hand, it is assumed that an attack is carried out by an intelligent malicious user [20] who adjusts the routes of the malicious packets as the controller changes flow rules to deny access to the node [20]. In this case, a specific node is targeted, and the C_A on the node's links will not decrease as flow rules are changed. Therefore, the node's $v_i^{l\max}$ will remain at an eigenindex i smaller than the phantom node's $v_i^{l\max}$ even after flow rules are changed to alleviate the congestion.

It is important to note that both an attack and a hardware failure cause a node's $v_i^{l\max}$ to remain at an eigenindex smaller than the phantom node's $v_i^{j\max}$; however, a hardware failure is distinguishable by the controller in a SDN through echo request and echo response messages. These messages act as the heartbeat for the individual nodes; if the controller experiences an echo request timeout, then it knows that it has lost connection to the node [32].

A second possible way to identify an attack before it completely takes down a node is through analysis of the rate at which $v_i^{l\max}$ shifts left in the eigenindex i. Generally, changes in traffic occur slowly [33], and the assumption made here is that an

attack causes changes in traffic at a much more rapid rate. This method requires the attack to occur at a rate dissimilar to normal congestion, the controller to poll the network or update the adjacency matrix at a frequency that detects the rate of change, and the remaining network to maintain a consistent state. This method is not evaluated in this thesis.

Overall, the concept of reference node allows the controller to identify and locate nodes that are congested, underutilized, or attacked. This information is then used by the controller in Phase 6 to produce outputs that help resolve congestion problems in the network.

6. Determine Outputs for Controller Action

Controller outputs are determined by the identification of congested, underutilized, and attacked nodes during network analysis. Three outputs are proposed in the monitoring scheme to provide information to either the routing function of the controller or to other applications that exist in the application layer of the SDN for the purpose of network management.

The first output is underutilization. This output provides the identification and location of the nodes that were determined to be underutilized during network analysis to the routing function of the controller. These nodes have enough capacity available to handle more packets across their links.

The second output is load balancing. This output provides the identification and location of nodes that are determined to be congested during network analysis to the routing function of the controller and requests load balancing for these nodes. The load balancing request tells the routing function to change flow rules to decrease the amount of traffic on the congested node's links.

The last output is isolation. This output provides the identification and location of nodes that are determined to be attacked during network analysis to an intrusion detection system (IDS) application in the SDN and requests the isolation of the malicious traffic. The IDS application is outside the scope of this thesis, but the controller must have a

packet parsing system, such as Snort [34], to verify the attack and isolate the malicious traffic from the malicious traffic; the isolation output acts as a tipper to the IDS to conduct further analysis and pass information to the routing function of the controller in order to isolate the source of the malicious traffic.

The details regarding how the routing portion of the controller uses the different outputs to update flow rules is beyond the scope of this thesis, but the outputs from this phase provide the controller with the necessary information to change flow rules for the purpose of reducing congestion or isolating an attack.

Together, the methods and processes employed in the six phases of the proposed monitoring scheme provide the controller with the identity and location of network disconnections, link failures, congested nodes, underutilized nodes, and attacked nodes. This information can be implemented into the flow rule generation process of the SDN controller to provide effective network management.

D. CONGESTION ESTIMATION METHOD

A proposed congestion estimation method to determine a node's capacity used when it crosses a congestion threshold in the proposed monitoring scheme is presented in this section. The amount of congestion required for a node to cross a congestion threshold is dependent upon the placement of the phantom node in the dual basis and the connectedness of the node when the network is at 100 percent capacity; a large amount of congestion is required for a node to shift influence from $v_i^{l\max}$ of a large eigenindex i to $v_i^{l\max}$ of the smallest i, while a smaller amount of congestion is required for a node to shift influence from $v_i^{l\max}$ of a small i to the smallest i. The estimation method is derived from algebraic connectivity boundaries and provides the user with a tool to determine the appropriate phantom node link weight value that identify nodal conditions at acceptable congestion levels.

1. Algebraic Connectivity Boundary Equations

Algebraic connectivity λ_2 has two bounds: an upper bound and a right bound. These bounds produce an envelope for λ_2 across the entire range of capacity used and describe the connectivity characteristics of the network. The equations of the two boundaries can be derived using known information about node connectivity k_N, link connectivity k_L, and the Laplacian matrix Q.

To determine the equation for the upper bound, a relationship between algebraic connectivity, node connectivity k_N, link connectivity k_L and the minimum degree $D_{\min}$ node is established. For an incomplete graph [13], [22],

$$k_N \le k_L \le D_{\min} \tag{23}$$

and [31]

$$\lambda_2 \le k_N, \tag{24}$$

which means

$$\lambda_2 \le D_{\min}. \tag{25}$$

For a complete graph [13], [22], [31]

$$k_N = k_L = D_{\min} \tag{26}$$

and [21]

$$\lambda_2 \le k_N, \tag{27}$$

which means

$$\lambda_2 \le D_{\min}. \tag{28}$$

A complete graph always has a higher algebraic connectivity λ_2 than an incomplete graph since the minimum degree $D_{\min}$ is larger [21]; therefore, the equation for algebraic connectivity of a complete graph is an upper bond for algebraic connectivity of incomplete graphs. This means the equation of the upper bound of algebraic connectivity λ_2^U is

$$\lambda_2^U = D_{\min}. \tag{29}$$

To determine the equation for the right bound, the eigenvalues of the weighted Laplacian matrix Q of a complete graph is analyzed. A complete graph provides maximum algebraic connectivity λ_2 since the minimum degree $D_{\min}$ is maximized [13], [21]. The link weights used in the weighted Laplacian are equal to capacity used C_U. Setting all link weights equal and solving for the eigenvalues, we obtain the characteristic equation [27]

$$(-1)^n \lambda(\lambda - n + C_U)^{n-2}(\lambda - n + nC_U) = 0 \tag{30}$$

for a weighted graph. Substituting λ_2 for λ into (30), we get the characteristic equation for algebraic connectivity λ_2 [27]

$$\lambda_2 = n(1 - C_U) \tag{31}$$

which can be rewritten as

$$\lambda_2 = (D_{\max} + 1)(1 - C_U) \tag{32}$$

since the total number of nodes n is equal to one more than the degree of the node in a complete graph that has maximum degree $D_{\max}$. Equations (31) and (32) cannot be used to determine algebraic connectivity λ_2 for an incomplete graph, but they provide a right bound since λ_2 is always larger for a complete graph [13], [21]. This means that the equation of the right bound of algebraic connectivity λ_2^R is

$$\lambda_2^R = (D_{\max} + 1)(1 - C_U). \tag{33}$$

An example of the algebraic connectivity boundaries defined by (29) and (33) are shown in Figure 11. The equations for the upper bound and right bound intersect at some capacity used C_U. Due to the derivation of (29) and (33), this intersection represents a switch in nodal influence in the second eigenindex; the intersection of the equations represents the point at which the node with maximum degree $D_{\max}$ becomes less connected than the node with minimum degree $D_{\min}$ and takes over primary nodal influence of λ_2. This is further illustrated in Chapter IV.

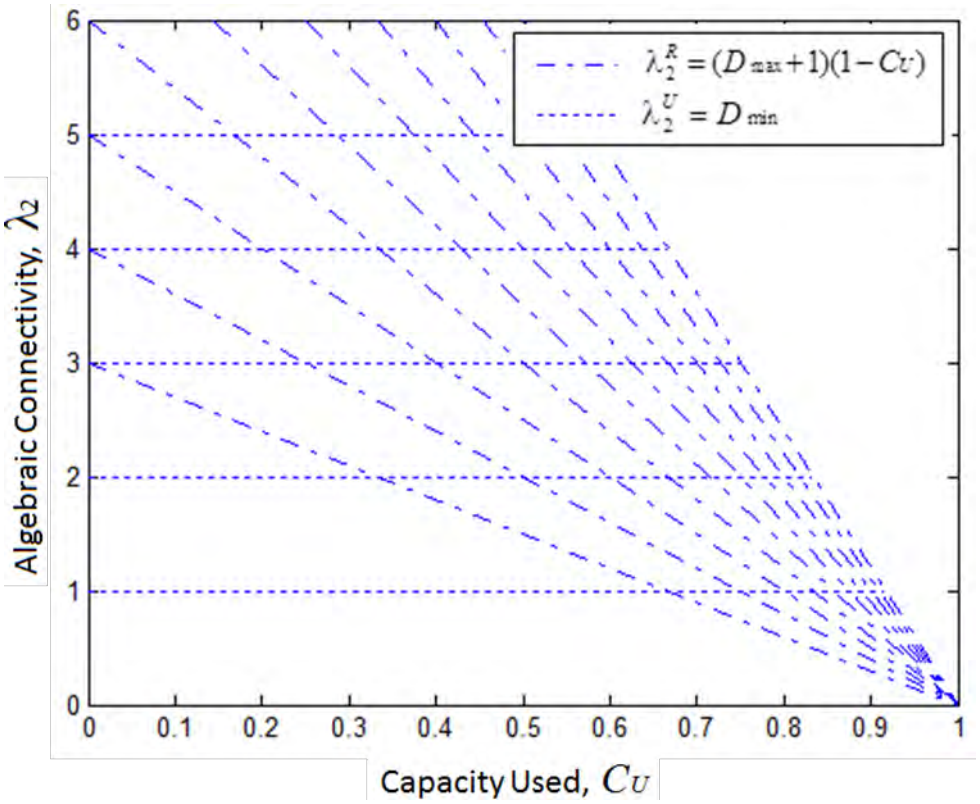


Figure 11. Example algebraic connectivity upper and right boundaries for networks with integer minimum and maximum degrees.

2. Estimating Congestion for Threshold Crossing

The equations for the upper and lower bounds on algebraic connectivity can be modified to estimate a node's capacity at the congestion threshold instead of estimating the maximum degree node at the threshold established by the minimum degree node, D_{min} in the dual basis, through substitution of the variables.

Substituting the minimum degree node D_{min} with the degree of the phantom node D_p in (29) yields

$$\lambda_2^U = D_p. \tag{34}$$

This modification sets the phantom node as the threshold instead of the minimum degree node; it is shown in Chapter IV that this substitution is valid even if the phantom node is not the node with minimum degree D_{min}. Substituting the maximum degree D_{max} with the degree of the node of interest D_I in (33) yields

$$\lambda_2^R = (D_l + 1)(1 - C_U). \tag{35}$$

This modification accounts for the all nodes in the network, not just the maximum degree node. Solving (34) and (35) for C_U at the point of intersection, C_{Uis}, we get

$$C_{Uis} = 1 - \frac{D_p}{(D_l + 1)} \tag{36}$$

which estimates a node's capacity used C_U at the instant it crosses the congestion threshold established by the phantom node with a given link weight value. If $C_{Uis} \geq 1$, then the node does not cross the congestion threshold until it fails, and if $C_{Uis} \leq 0$, then the node always has influence over a smaller eigenvalue compared to the phantom node.

Overall, (36) provides users with a congestion estimation method that can be used to determine the appropriate phantom node link weight value to correctly position the phantom node to achieve desired monitoring results in the proposed monitoring scheme. Depending on the type of network and its requirements, we see that more or less C_U may be allowed or desired on a node before being flagged as congested. A larger allowed C_U means that the controller is less active because threshold crossings are infrequent; however, threshold crossings then only occur when the node is close to failure. On the other hand, a smaller allowed C_U means that the controller is more active because threshold crossings occur more frequently; but the crossings occur long before node failure.

A network monitoring scheme that uses graph theory analysis and phantom nodes to produce outputs that are used by the controller to update flow rules and manage a network was proposed in this chapter. The outputs identify disconnected, congested, underutilized, and attacked nodes by comparing the node's health and connectedness to the health and connectedness of the phantom node in the dual basis [17], [23]. The position of the phantom node in the dual basis is determined by its link weight value and a congestion estimation method was proposed to help identify the appropriate link weight value to achieve desired monitoring results. Overall, the proposed monitoring scheme provides the centralized controller with a novel method to monitor and manage large,

complex networks. These analysis techniques, along with the congestion estimation method, are simulated in the next chapter to determine the validity and accuracy of the proposed monitoring scheme.

IV. SIMULATION AND RESULTS

A SDN monitoring scheme for determining network connectivity, congestion, underutilization and attack was proposed in Chapter III. The functions and methods required to implement the monitoring scheme are validated in this chapter through MATLAB simulation. First, the methodologies are discussed for producing random networks, creating graph theory matrices, and implementing congestion; these methodologies are used in all the simulations in this chapter. Then, the connectivity simulations and results are discussed, including a disconnected network simulation and a link failure simulation. Next, the reference node concept and the method for identifying congestion, underutilization, and attack are simulated. Lastly, the method for estimating nodal congestion for a congestion threshold crossing is evaluated.

A. METHODOLOGY

The purpose of the simulations in this chapter is to show that the controller is able to use the graph theory analysis to monitor node health and identify disconnected, congested, underutilized and attacked nodes. All simulations use the same method to create random networks and model dynamic changes in the network using MATLAB version 2013b.

First, all simulations require the creation of a random network in the form of an adjacency matrix. The simulations in this chapter use open source MATLAB code to produce the adjacency matrix A of a random modular network [35]. The open source code requires four inputs to produce the adjacency matrix A: number of nodes n, number of clusters or modules c, overall probability of attachment p, and proportion of links within modules r. For inputs of $n = 6$, $c = 2$, $p = 0.3$, and $r = 0.3$, the open source code created

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{37}$$

The ones in (37) indicate a link between the i^{th} and j^{th} nodes, as defined by (4), while the zeroes indicate that a link does not exist. This information is used to create the network topology displayed in Figure 12. It is important to note that the same input values produce a different random network every time the code runs.

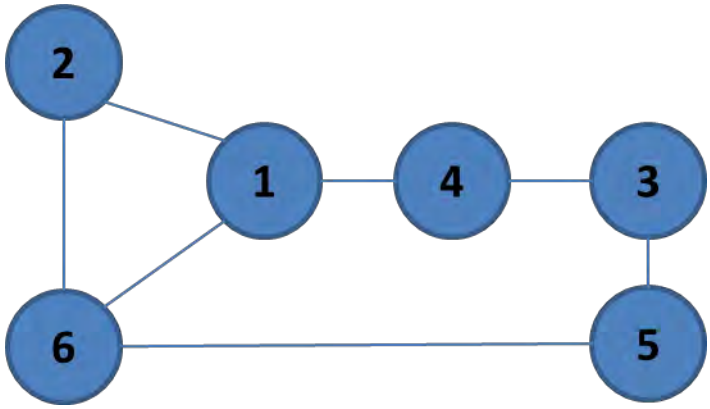


Figure 12. Network topology of a random six-node modular network with two modules and an overall probability of attachment of 0.3.

Second, all simulations require the creation of the eigenvalue matrix Λ and the eigenvector matrix V for network analysis. The *eig* command in MATLAB is used to produce these matrices. The command, $[V, D] = eig(Q)$, solves (8) and produces a diagonal eigenvalue matrix D and an eigenvector matrix V [36]. Since the eigenvector matrix V contains elements with negative magnitudes, all simulations transform V into V_s using the procedure explained in Chapter III to produce a functional matrix for network analysis.

Lastly, all simulations require a method for varying link capacity to produce link failures or congestion in order to simulate a dynamic network. The elements of the adjacency matrix A are equal to the link's capacity available C_A. These elements are modified during the simulations to produce varying levels of congestion and connectivity. It is important to note that the C_A value is never modified during a simulation for the link attached to the phantom node.

The methodology presented provides a repeatable process that can be used across a large number of random networks. The process is used in each simulation to produce results that demonstrate the effectiveness of the graph theory analysis methods of the proposed network monitoring scheme. The results of the connectivity simulations are presented in the next section.

B. CONNECTIVITY RESULTS

The simulation results are presented for the disconnected node case and the link failure case to validate the connectivity analysis method used in the proposed monitoring scheme. The disconnected node simulation demonstrates that disconnected nodes can be identified and located using the null space, and the link failure simulation confirms that neither algebraic connectivity λ_2 nor the null space can be used to identify link failures.

1. Disconnected Nodes Simulation

The null space provides all the required information to identify and locate a disconnected node(s) within a network. To illustrate this concept, a network was simulated to experience three random node disconnections. This simulation was conducted 25 times to ensure the accuracy of the results. Since all of the results were similar, the results of only one run are presented here. This run used inputs of $n = 8$, $c = 2$, $p = 0.4$, and $r = 0.35$ [35], which produced an adjacency matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{38}$$

The disconnections in the network topology that occurred during the simulation are shown in Figure 13. First, the initial eigenvector matrix V and eigenvalue matrix Λ are computed using the network topology shown in Figure 13(a). Then, node 1 is disconnected and the matrices are recalculated based on the new adjacency matrix; The network topology after node 1 is disconnected is shown in Figure 13(b). This process is then repeated as node 6 is disconnected and node 4 is disconnected; the network topology after these disconnections is shown in Figure 13(c) and Figure 13(d). This simulation produced a set of Λ and V outputs for each of the four steps.

The eigenvalue and eigenvector matrices of the random network in Figure 13(a) show that the null space correctly identifies a connected network. The eigenvalues produced during this step are $\{0, 1.6425, 1.7828, 3.4087, 3.7256, 4.9399, 6.0376, 6.4629\}$, and the associated eigenvector matrix output V_a is

$$V_a = \begin{bmatrix} 0.0000 & 0.0072 & 0.0509 & 0.0083 & 0.0278 & 0.1392 & 0.1623 & 0.4794 \\ 0.0000 & 0.0312 & 0.0240 & 0.0810 & 0.3510 & 0.0074 & 0.1552 & 0.1551 \\ 0.0000 & 0.1787 & 0.5606 & 0.0008 & 0.0027 & 0.1005 & 0.0159 & 0.0157 \\ 0.0000 & 0.0011 & 0.0809 & 0.2360 & 0.0418 & 0.2079 & 0.2805 & 0.0268 \\ 0.0000 & 0.0064 & 0.0013 & 0.2888 & 0.0140 & 0.1789 & 0.3189 & 0.0066 \\ 0.0000 & 0.0766 & 0.2015 & 0.0092 & 0.4550 & 0.0307 & 0.0518 & 0.0501 \\ 0.0000 & 0.6274 & 0.0410 & 0.0431 & 0.1068 & 0.0062 & 0.0124 & 0.0381 \\ 0.0000 & 0.0114 & 0.0396 & 0.3329 & 0.0008 & 0.2591 & 0.0030 & 0.2282 \end{bmatrix}. \tag{39}$$

From (12), $I_z = 1$ for (39) which means $N_d = 0$, where N_d is the total number of disconnected nodes [14], [21]. This information shows that the network is connected [21].

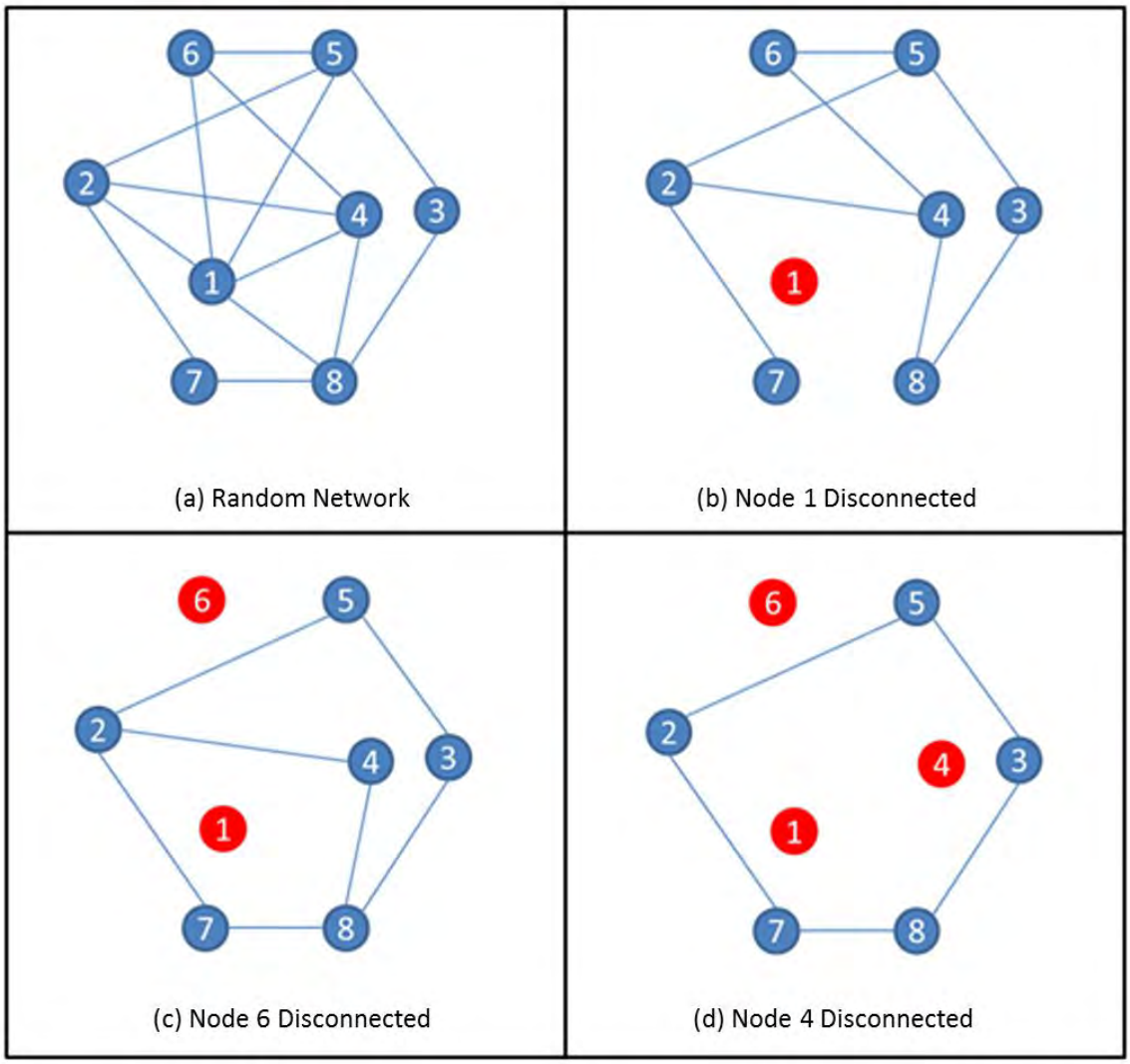


Figure 13. Network topology showing the order of node disconnections during the node disconnections simulation.

After node 1 is disconnected, the eigenvalue and eigenvector matrices show that the null space correctly identifies the disconnected node. The resultant eigenvalues are $\{0, 0, 1.3249, 1.5858, 2.4608, 3, 4.4142, 5.2143\}$, and the associated eigenvector matrix output V_b is

$$V_b = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0148 & 0.0788 & 0.1996 & 0.2500 & 0.0283 & 0.2856 \\ 0.0000 & 0.0000 & 0.0000 & 0.6306 & 0.0000 & 0.0000 & 0.2265 & 0.0000 \\ 0.0000 & 0.0000 & 0.0148 & 0.0788 & 0.2500 & 0.2500 & 0.0283 & 0.2856 \\ 0.0000 & 0.0000 & 0.0912 & 0.0270 & 0.0000 & 0.0000 & 0.3301 & 0.1355 \\ 0.0000 & 0.0000 & 0.3940 & 0.0788 & 0.2500 & 0.2500 & 0.0283 & 0.0788 \\ 0.0000 & 0.0000 & 0.3940 & 0.0788 & 0.2500 & 0.2500 & 0.0283 & 0.0788 \\ 0.0000 & 0.0000 & 0.0912 & 0.0270 & 0.0000 & 0.0000 & 0.3301 & 0.1355 \end{bmatrix}. \quad (40)$$

This time, $I_z = 2$ for (40) which means $N_d = 1$ from (12); one disconnected node N_d exists in the network [14], [21]. Analyzing the null space, we see that $v_1^1 = 1$, which means node 1 is responsible for $\lambda_1 = 0$ and disconnected from the network [21].

After node 1 and node 6 are disconnected, the eigenvalue and eigenvector matrices show that the null space correctly identifies the disconnected nodes. The resultant eigenvalues are $\{0,0,0,1.2679,2,2,4,4.7321\}$, and the associated eigenvector matrix output V_c is

$$V_c = \begin{bmatrix} 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0223 & 0.2440 & 0.0060 & 0.2500 & 0.3110 \\ 0.0000 & 0.0000 & 0.0000 & 0.3110 & 0.2440 & 0.0060 & 0.2500 & 0.0223 \\ 0.0000 & 0.0000 & 0.0000 & 0.1667 & 0.0121 & 0.4879 & 0.0000 & 0.1667 \\ 0.0000 & 0.0000 & 0.0000 & 0.3110 & 0.2440 & 0.0060 & 0.2500 & 0.0223 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.1667 & 0.0121 & 0.4879 & 0.0000 & 0.1667 \\ 0.0000 & 0.0000 & 0.0000 & 0.0223 & 0.2440 & 0.0060 & 0.2500 & 0.3110 \end{bmatrix}. \quad (41)$$

This time, $I_z = 3$ for (41), which means $N_d = 2$ from (12); two disconnected nodes N_d exists in the network [14], [21]. Analyzing the null space, we see that $v_1^6 = 1$ and $v_2^1 = 1$, which means node 6 is responsible for $\lambda_1 = 0$ and node 1 is responsible for $\lambda_2 = 0$. This information identifies nodes one and six as the disconnected nodes in the network [21].

Lastly, after node 1, node 4, and node 6 are disconnected, the eigenvalue and eigenvector matrices show that the null space correctly identifies the disconnected nodes. The resultant eigenvalues are $\{0,0,0,0,1.3820,3.6180,3.6180\}$, and the associated eigenvector matrix output V_d is

$$V_d = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.2410 & 0.1590 & 0.3618 & 0.0382 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0265 & 0.3735 & 0.1382 & 0.2618 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.2819 & 0.1181 & 0.3618 & 0.0382 \\ 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0518 & 0.3482 & 0.1382 & 0.2618 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.3988 & 0.0012 & 0.0000 & 0.4000 \end{bmatrix}. \quad (42)$$

This time, $I_z = 4$ for (42), which means $N_d = 3$ from (12); three disconnected nodes N_d exist in the network [14], [21]. Analyzing the null space, we get $v_2^6 = 1$, $v_3^4 = 1$ and $v_4^1 = 1$, which means node 6 is responsible for $\lambda_2 = 0$, node 4 is responsible for $\lambda_3 = 0$, and node 1 is responsible for $\lambda_4 = 0$. This information identifies nodes 1, 4, and 6 as disconnected nodes in the network [21].

The results show that the null space provides the necessary information to determine the location and number of disconnected nodes in a network. The null space provides information about nodal connectivity but does not provide information regarding link connectivity, which is the focus of the next connectivity simulation.

2. Link Failure Simulation

The proposed network monitoring scheme uses the link weights of the adjacency matrix A to identify link failures because algebraic connectivity and the null space do not provide the information required [13], [23]. To demonstrate this capability, 25 networks were simulated to experience link failures on individual nodes. The inputs $n = 15$, $c = 3$, $p = 0.5$, and $r = 0.5$ were used to create the 25 random adjacency matrices [35]. Each node was then categorized according to its degree, and the link

weights were reduced from one to zero. After each link reduction, algebraic connectivity λ_2 was recalculated. The algebraic connectivity results were averaged and plotted according to the degree of the node to determine the effects of link failures on algebraic connectivity.

The averaged algebraic connectivity results from the simulations are displayed in figure 14. The average algebraic connectivity of the network is approximately 3.25 before any links on the four degree nodes were reduced but drops to 2.5, then 1.75, then one, and lastly zero as each of the four links fail.

From the results shown in Figure 14, the null space does not identify individual link failures. For each set of nodes with the same degree, algebraic connectivity λ_2 reaches zero when all of the links are fully utilized. This means a node does not enter the null space until all of its links have failed; a single link failure cannot be detected in the null space.

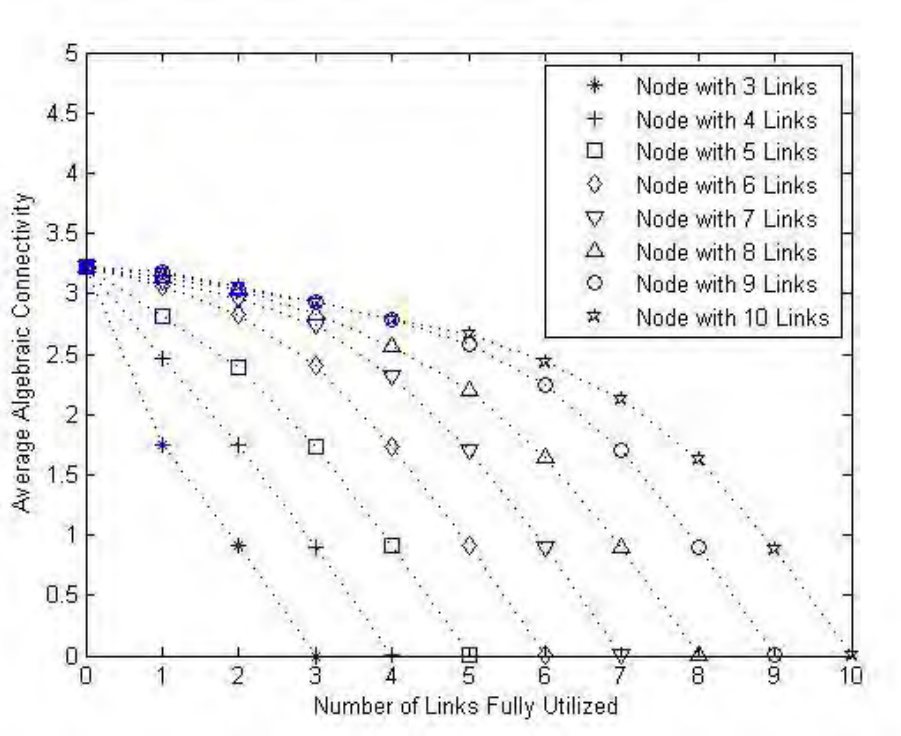


Figure 14. Algebraic connectivity of 15-node networks as individual link weights are reduced from one to zero. The plots are obtained by averaging the results of 25 random networks.

Analysis of the results also shows that algebraic connectivity is not an effective tool for identifying link failures. The curves for each degree node have a distinct bend when approximately 50 to 60 percent of a node's links fail for nodes that have a degree greater than four [27]. This means that algebraic connectivity is relatively unaffected by link failures until over 50 percent of its links have failed. For nodes with a degree less than four, algebraic connectivity is affected because the node is already the minimum degree $D_{\min}$ node in the network and primary influencer of λ_2.

Additionally, the results show that we cannot determine the location of the link failures based on a reduction in algebraic connectivity. For example, the average algebraic connectivity is approximately 1.75 for a five degree node with three fully utilized links. It is also approximately 1.75 for a six degree node with four fully utilized links. This means a link failure on two different degree nodes could be responsible for the algebraic connectivity value, making it impossible to correctly identify the location of a link failure using algebraic connectivity.

The disconnected node simulation showed that the null space is useful to determine node connectivity, and the link failure simulation showed that the null space and algebraic connectivity are not useful in determining link connectivity. The results from both simulations validate the methods used to perform connectivity analysis in the proposed monitoring scheme. The results of the congestion simulations to show the accuracy of the proposed methods for identifying nodal conditions are presented in the next section.

C. CONGESTION RESULTS

Simulation results are presented in this section to validate the congestion analysis methods used in the proposed monitoring scheme. The first simulation validates the concept of determining tracking a node's health using the dual basis. The second simulation validates the phantom reference node concept and congestion threshold concept for identifying nodal states. The results from these two simulations validate the graph theory analysis methods used in the proposed monitoring scheme to provide congestion analysis.

1. Node Health Tracking

The dual basis provides all the required information to track the changes in a node's health and connectedness over time. To illustrate this concept, a network was simulated to experience congestion on one node for a duration of two seconds. The simulation was conducted 30 times to ensure the accuracy of the results. Since all of the results were similar, the results of only one run are presented here. This run used

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{43}$$

which was produced for $n = 8$, $c = 2$, $p = 0.45$, and $r = 0.35$ [35]. The resultant network topology formed by (43) is displayed in Figure 15.

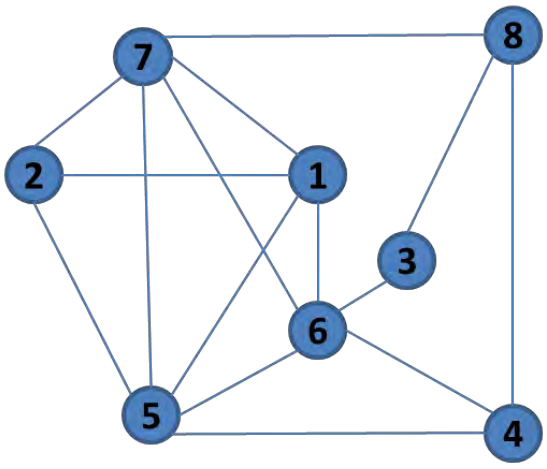


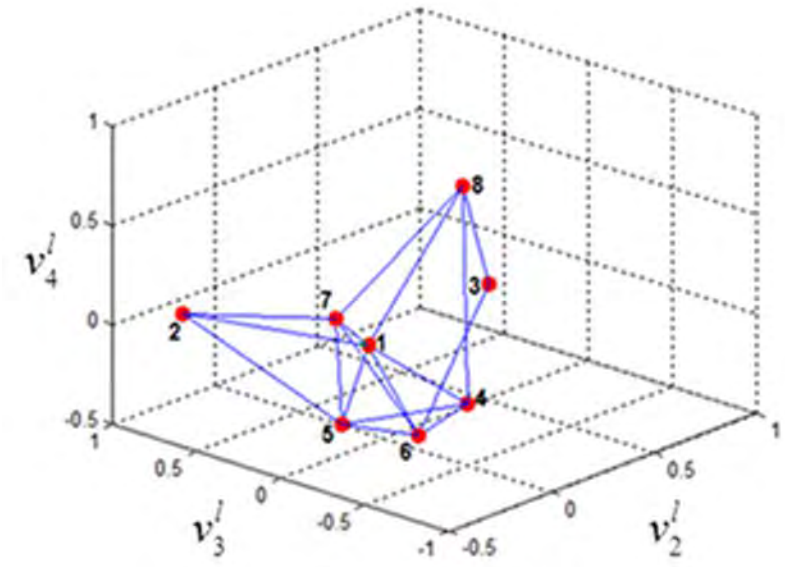
Figure 15. Network topology of random modular network formed by the eight node adjacency matrix with two modules and an overall probability of attachment of 0.45.

Node 1 was selected as the node to experience the congestion because it is one of the most connected nodes in the network and has $v_i^{l\max}$ in a large eigenindex i. This was determined by plotting the network centrality graph [29] and the nodal influence graph as shown in Figure 16. In the network centrality graph shown in Figure 16(a), node 1 is plotted near the middle of the graph, giving it a large network centrality metric [30]. Additionally, node 1 has large nodal influences on the seventh and eighth eigenindices i compared to its nodal influence on the remaining eigenindices. While node 7 had a larger nodal influence in the seventh and eighth eigenindices, node 1 was selected because it had a larger centrality metric.

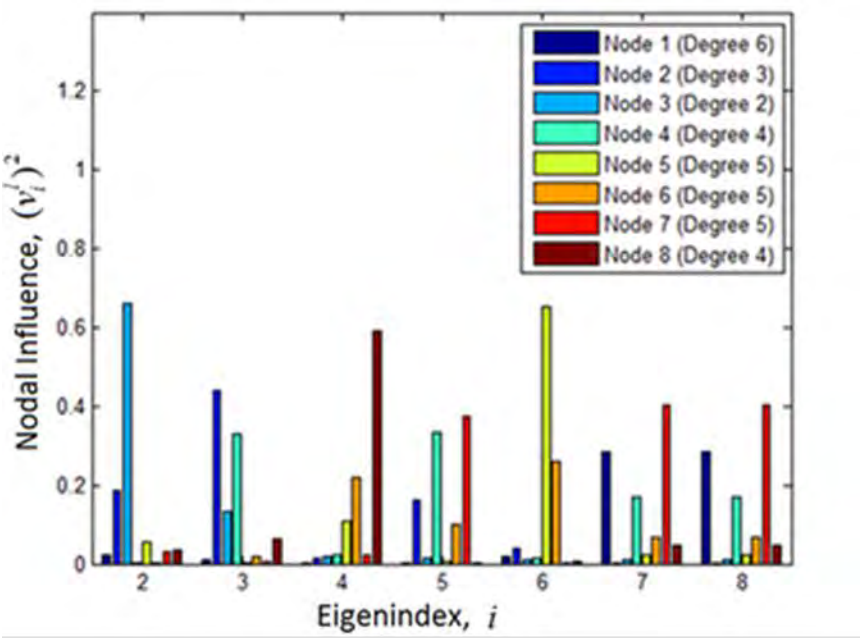
Then, all of node 1's link weights were decreased in 0.01 increments; in two seconds, all of the link weights were decreased from one to zero. After each incremental decrease, the eigenvector matrices V were recalculated from the new adjacency matrix A. The eigenvector results for each node were then compiled together to determine if the dual basis tracked changes in node health and connectivity.

As shown in Figure 17, the nodal influence v_i^l are plotted as a function of time for each node across all eight eigenindices i. Node 1 initially has primary nodal influence $v_i^{l\max}$ in both the seventh and eighth eigenindices, and node 5 has $v_i^{l\max}$ in the sixth eigenindex. At approximately 0.25 seconds, node 1 takes over $v_i^{l\max}$ in the sixth eigenindex and node 5 becomes the primary nodal influencer of the seventh and eighth eigenindices. This shift in nodal influence within the eigenindex means that the congestion on node 1's links has made it weaker or less connected than node 5 [17], [23].

As congestion grows, node 1 becomes weaker than nodes 4 and 7 at approximately 0.7 seconds and takes over primary nodal influence $v_i^{l\max}$ in the fifth eigenindex. At the same time, nodes 4 and 7 take over $v_i^{l\max}$ in the sixth eigenindex. This shift in nodal influence within the eigenindex i continues all the way until node 1 takes over $v_i^{l\max}$ of the second eigenindex at approximately 1.6 seconds. At this point, node 1 is the weakest and least connected node in the network due to the congestion on its links.



(a) Network Centrality



(b) Nodal Influence

Figure 16. (a) Network centrality of random modular eight-node network using second, third, and fourth eigenindices (b) Nodal influence plots of all nodes of the random modular eight-node network across all eigenindices, after [29].

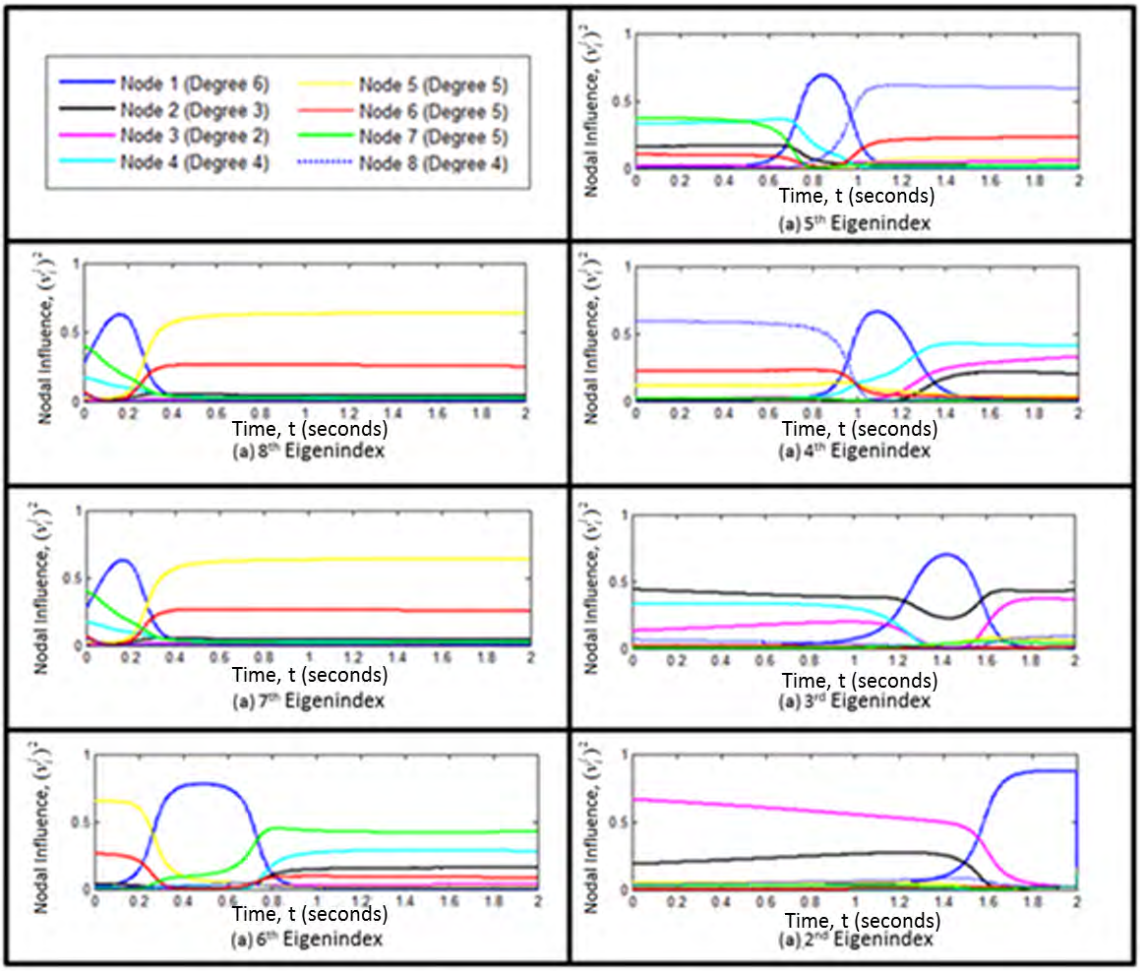


Figure 17. Nodal influence plots for all eight eigenindices of a random modular eight node network as capacity used is increased on all links of the maximum degree node (node 1) from zero to one.

The results illustrate that the dual basis, specifically the nodal influence, can be used to determine a node's health relative to other nodes in the network. Additionally, the shift in nodal influence within the eigenindex i can be monitored to determine changes in nodal health due to network conditions. This means that a controller can track changes in a node's health and connectedness over time. This concept is applied in the next simulation to show that a phantom node establishes a congestion threshold in the dual basis, and a node's state can be identified using the congestion threshold.

2. Node Health Identification

A phantom node establishes a congestion threshold that is used to determine if a node is congested, underutilized, or attacked. To illustrate this concept, congestion is added to three links for three seconds in a network that has a single phantom node. Additionally, the simulation keeps the link weights of a node constant once the node fails to model an unresponsive or attacked node.

The simulation was conducted on 50 random networks of various sizes to ensure the accuracy of the results. Since all of the results were similar, the results of only one run are presented here to demonstrate the concept. In this run, the simulation used (43) as the initial adjacency matrix A. Then a phantom node was attached to node 7 with a link weight of equal to one, which changed (43) to

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \quad (44)$$

Node 7 was selected for phantom node attachment because it is one of the most central nodes in the network. The phantom node is node 9 in (44) and has a link weight of one. The resultant network topology formed by (44) is displayed in Figure 18. The phantom node is designated by the green node, and a green dotted link is used to show that the phantom node does not exist in the physical topology and only exists in the adjacency matrix.

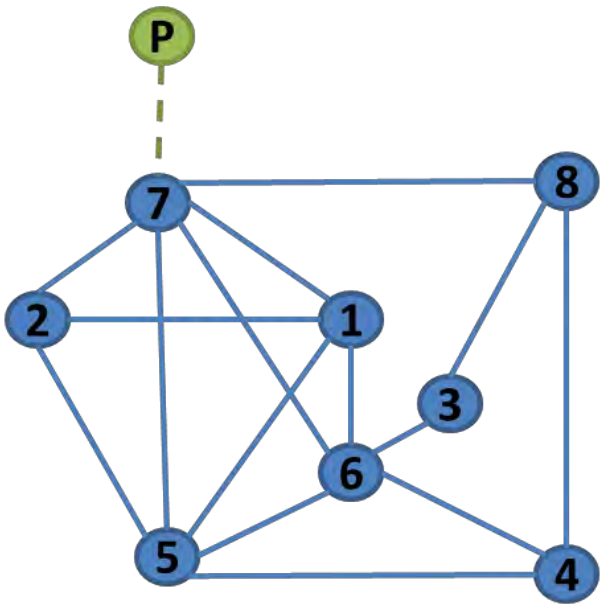


Figure 18. Network topology of an eight node network containing one phantom node (designated as P) attached to node 7.

Next, three random nodes were selected to experience congestion. Node 1's link weights were decreased from one to zero between zero and one seconds to simulate congestion and then remained zero from one to three seconds to simulate unresponsiveness to changes in flow rules. Node 3's link weights were decreased from one to zero between one and two seconds to simulate congestion and then remained zero from two to three seconds to simulate an unresponsive node. Lastly, node 4's link weights were decreased from two to three seconds to simulate congestion. All of the link weights were decreased in 0.01 increments, and the eigenvector matrices were recalculated after each incremental decrease from the new adjacency matrix. The eigenvector results were compiled together to produce nodal influence plots. These plots are used to first determine if the phantom node established a threshold that can be used to identify the health of the node and then determine if the node can be identified the instant it crosses the congestion threshold.

a. Congestion Threshold Establishment

To validate the concept of the phantom node establishing a congestion threshold and if a node's health can be identified using the congestion threshold, the nodal basis was plotted for all nodes in the network at four instants in time, as shown in Figure 19. The nodal influence $(v_i^l)^2$ of all nodes at network capacity available $C_A = 1$ is shown in Figure 19(a). The phantom node has primary nodal influence $v_2^{l\max}$, node 1 has $v_8^{l\max}$, node 3 has $v_3^{l\max}$, and node 4 has $v_6^{l\max}$. In other words, the phantom node has established a congestion threshold at $i = 2$ by having $v_2^{l\max}$ at the start of the simulation.

The nodal influence $(v_i^l)^2$ of all nodes at one second is shown in Figure 19(b). At this point, node 1's link weights are all equal to zero, and a shift in nodal influence v_i^l has occurred in the eigenindex i spectrum. Now, the phantom node has primary nodal influence $v_3^{l\max}$, node 1 has $v_2^{l\max}$, node 3 is one of the two primary nodal influencers of eigenindices $i = 4$ and $i = 5$, and node 4 has primary nodal influence in eigenindices $i = 5$, $i = 7$, and $i = 8$. The shift of nodal influence by the phantom node from $i = 2$ to $i = 3$ indicates a threshold crossing. Node 1 has primary nodal influence of a smaller eigenindex i compared to the phantom node, which means that node 1 has crossed the congestion threshold at some point during the first second of the simulation. Additionally, the plot shows that node 1 has $(v_2^l)^2 = 1$ and $(v_i^l)^2 = 0$ for all remaining i^{th} eigenvectors because it has entered the null space and is disconnected from the network.

The nodal influence $(v_i^l)^2$ of all nodes at two seconds is shown in Figure 19(c). At this point, node 3's link weights are now all equal to zero, node 1's link weights remain zero, and another shift in nodal influence has occurred in the eigenindex i spectrum. Now, the phantom node has primary nodal influence $v_4^{l\max}$, node 1 has $v_3^{l\max}$, node 3 has $v_2^{l\max}$, and node 4 has $v_7^{l\max}$. Both nodes 1 and 3 are primary nodal influencers of smaller eigenindices i compared to the phantom node. This means that node 3 crossed the congestion threshold at some point between one and two seconds, and node 1 remained

above the congestion threshold, which indicates that it was unresponsive to changes in flow rules.

The nodal influence $(v_i^l)^2$ of all nodes at three seconds is shown in Figure 19(d). At this point, node 4's link weights are now all equal to zero, the link weights of nodes 1 and 3 remain zero, and another shift in nodal influence has occurred in the eigenindex i spectrum. Now, the phantom node has primary nodal influence $v_5^{l\max}$, node 1 has $v_4^{l\max}$, node 3 has $v_2^{l\max}$, and node 4 has $v_3^{l\max}$. Nodes 1, 3, and 4 all have primary nodal influence in a smaller eigenindices i compared to the phantom node; node 4 crossed the congestion threshold at some point between two and three seconds, and both nodes 1 and 3 remain unresponsive at a congestion level above the congestion threshold.

These results demonstrate that the phantom node establishes a congestion threshold within the dual basis, and the shifts in nodal influence $(v_i^l)^2$ across the eigenindex i spectrum identify changes in nodal connectivity and congestion relative to the phantom node. Next, the results are analyzed for congestion identification when the node crosses the threshold established by the phantom node.

b. Node Congestion Identification at Threshold Crossing

To validate the concept of a node being identified the instant it crosses the congestion threshold, the changes in nodal influence $(v_i^l)^2$ for $i=2$, $i=3$, and $i=4$ were plotted for the three-second duration of the simulation, as shown in Figure 20. In all the plots, each line represents the nodal influence $(v_i^l)^2$ of an individual node in that particular eigenindex i. The node that has the largest nodal influence $v_i^{l\max}$ is the primary influencer of the eigenvalue λ_i associated to the eigenindex i [17], [21]. The discontinuous jumps in the second and third eigenindices observed at $t=1$ and $t=2$, respectively, are the result of the eigenvector being formatted into the standard eigenvector matrix V_s when a node enters the null space.

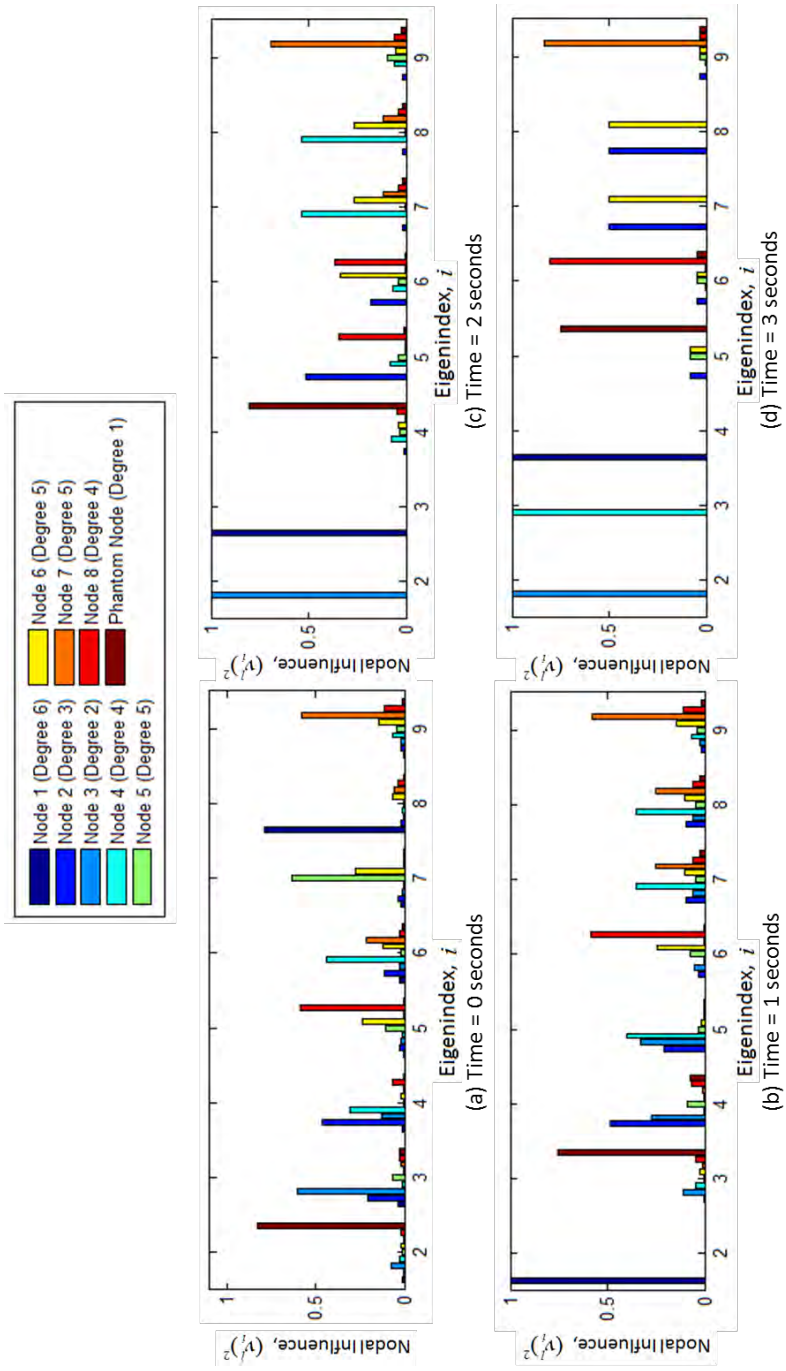


Figure 19. (a) Initial nodal influence plots of all nodes of a random modular eight-node network with two modules and an overall probability of attachment of 0.45; (b) Nodal influence plots at one second, after node 1's link weights are reduced from one to zero; (c) Nodal influence plots at two seconds, after node 3's link weights are reduced from one to zero; and (d) Nodal influence plots at three seconds, after node 4's link weights are reduced from one to zero.

Three distinct congestion threshold crossings are shown in Figure 20. The first threshold crossing occurs at $t \approx 0.87$ seconds in the second eigenindex when node 1 is approximately 87 percent congested; node 1 takes over $v_2^{l\max}$ and the phantom node takes over $v_3^{l\max}$. The second threshold crossing occurs at $t \approx 1.52$ seconds in the third eigenindex when node 3 is approximately 52 percent congested; node 3 takes over $v_3^{l\max}$ and the phantom node takes over $v_4^{l\max}$. The last threshold crossing occurs at $t \approx 2.69$ seconds when node 4 is approximately 69 percent congested; node 4 takes over $v_4^{l\max}$ and the phantom node takes over $v_5^{l\max}$, which is not shown.

Using the results of the congestion threshold simulation, we validated the method for identifying congested, underutilized and attacked nodes since a the phantom node establishes a congestion threshold, and a node can be identified in the dual basis as crossing the threshold when it becomes congested. This means that a node is identified as congested if it has primary nodal influence $v_i^{l\max}$ over a smaller eigenindex i compared the phantom node. Conversely, the results show that nodes 5, 6, and 7 consistently maintain primary nodal influence $v_i^{l\max}$ over the largest eigenindices. These nodes have large degrees, as shown in Figure 18, and experience zero congestion. This means that a node is identified as underutilized if it has primary nodal influence $v_i^{l\max}$ of an eigenindex i much larger than the phantom node. Lastly, the results show that a node is identified as attacked if it remains primary influencer of a smaller eigenindex i compared to the phantom node even after measures have been taken to reduce the congestion. This leaves the accuracy of congestion estimation method to be determined, which is the focus of the simulations in the next section.

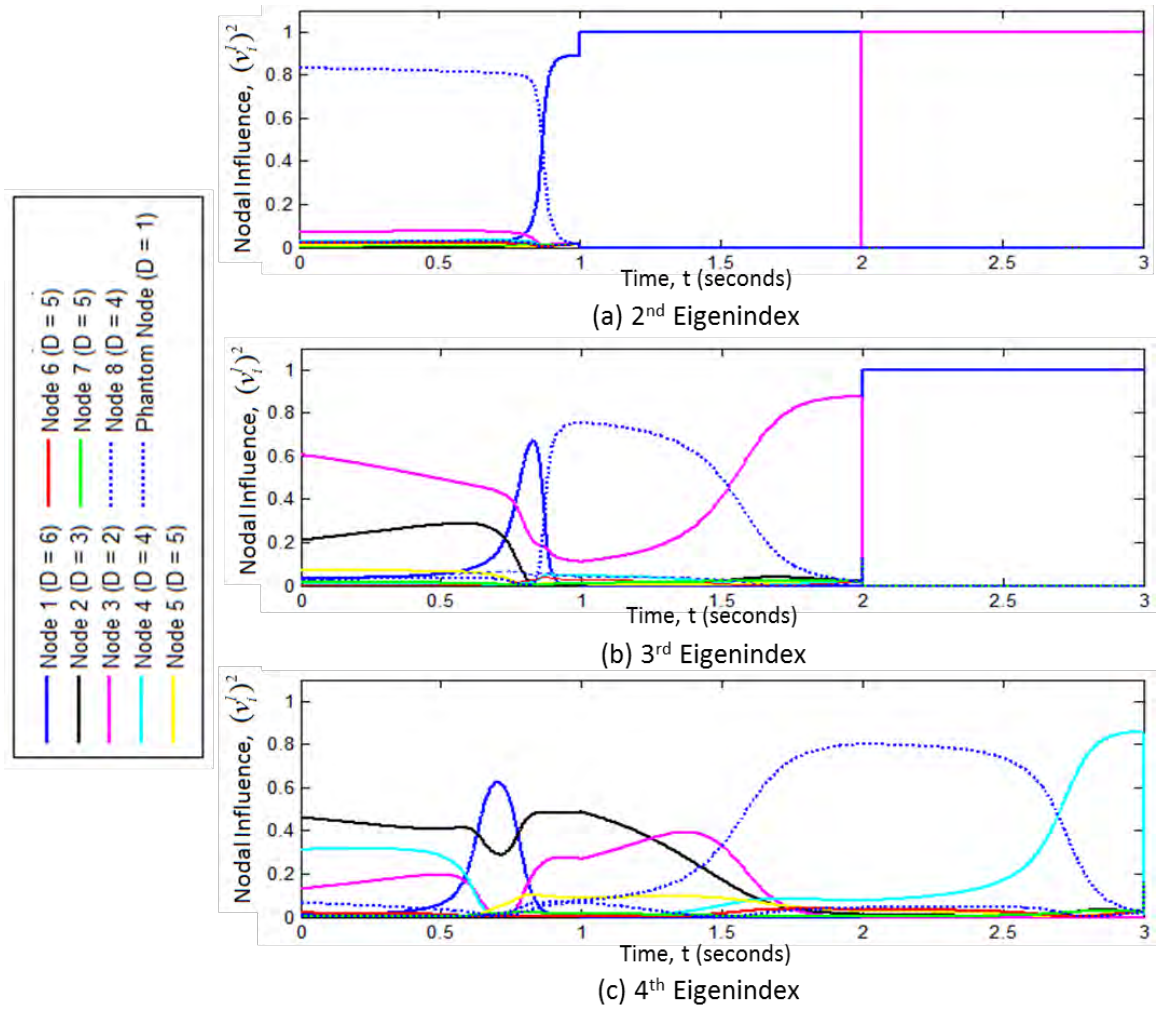


Figure 20. Nodal influences in the second, third, and fourth eigenindices as node 1's links go to zero from zero to one second, node 3's links go to zero from one to two seconds, and node 4's links go to zero from two to three seconds.

D. CONGESTION ESTIMATION METHOD

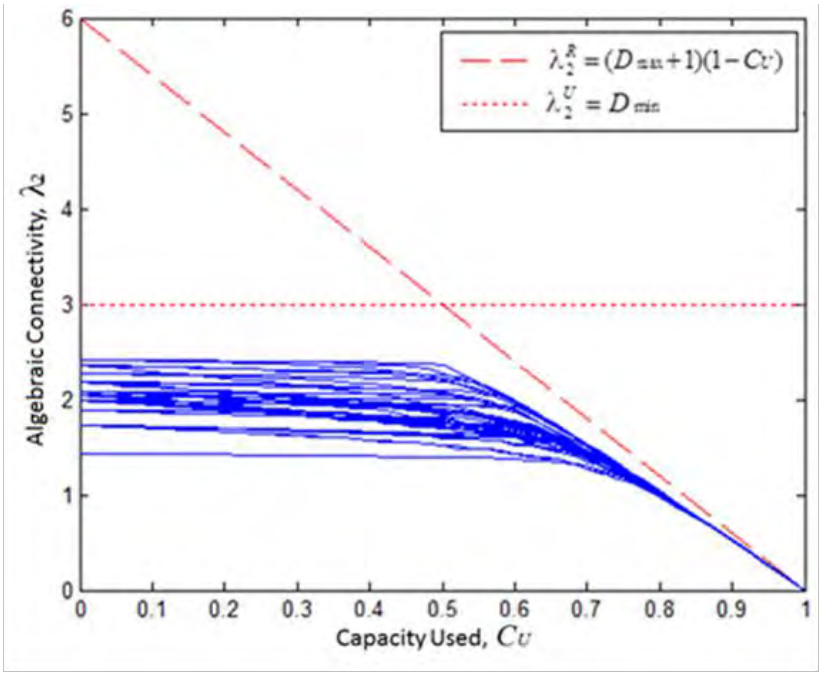
The congestion estimation method provides users with a tool to determine the amount of congestion a given phantom node link weight value allows on a node. To illustrate this concept, two simulations are presented. First, networks are simulated to experience congestion to validate the algebraic connectivity λ_2 boundary equations from Chapter III that provide the foundation for the estimation method. Then, congestion estimation simulations are presented to show that the estimation method is accurate in determining nodal congestion when it crosses a congestion threshold.

1. Algebraic Connectivity Boundary Equations

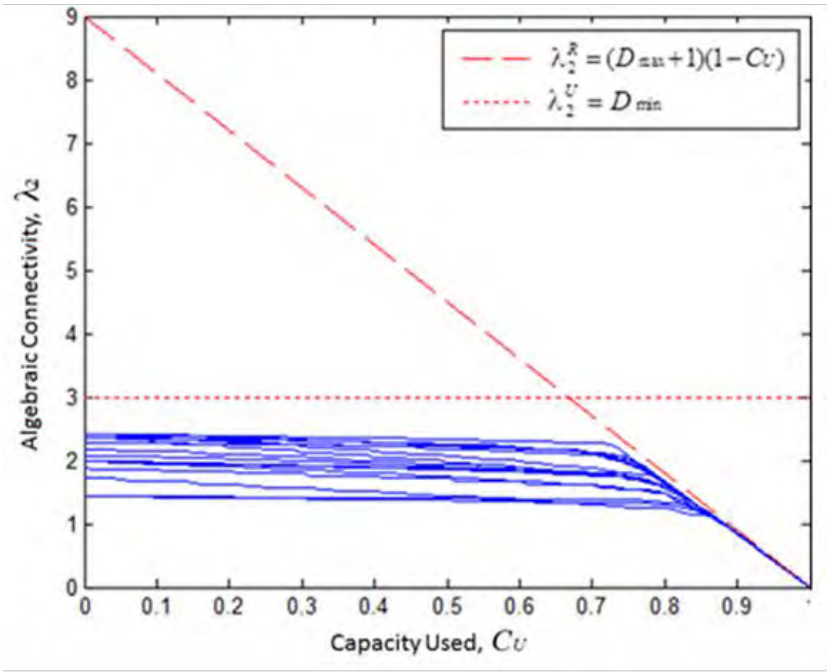
The boundary equations for algebraic connectivity not only bound algebraic connectivity but also provide an estimation of nodal capacity when the node takes over as primary nodal influencer of λ_2. To illustrate the validity of (29), (33), and (35), congestion was simulated across 30 random 15-node networks that each had a minimum degree $D_{\min} = 3$. Each node was categorized according to degree, and the link weights were incrementally decreased from one to zero. Algebraic connectivity λ_2 was recalculated and recorded after each decrease in link weight. The results were compiled together according to the degree of the node to determine the accuracy of the algebraic connectivity boundary equations and if the intersection point provides an estimate for when a node takes over primary nodal influence $v_2^{j\max}$.

The simulation was conducted 15 times and the results were similar regardless of the degree of the node of interest D_l. Only the results of two different degree nodes, $D = 5$ and $D = 8$, from one simulation are presented here. These results are displayed in Figure 21. The recorded changes in algebraic connectivity are shown by the blue lines, and the algebraic connectivity boundary equations are plotted using red dashed and dotted lines. Both plots show that the network's algebraic connectivity λ_2 always falls within the operating envelope formed by the boundary equations. The equations accurately provide upper and right bounds for λ_2.

Additionally, the plots shown in Figure 21 demonstrate that algebraic connectivity λ_2 stays relatively unchanged until the node's capacity reaches the intersection of the two boundary equations. After the intersection, the results show a rapid decrease in λ_2. This outcome is consistent with the results from the node health tracking simulation in Section C; λ_2 is not affected by congestion unless the node that has $v_2^{l\max}$ is experiencing congestion. This means that the intersection of the two algebraic connectivity boundary equations provides an estimate for a node's C_U when it takes over $v_2^{l\max}$. This concept is the foundation for the congestion estimation method, which is the focus of the next simulation.



(a) $D_{min} = 3$ and $D_I = 5$



(b) $D_{min} = 3$ and $D_I = 8$

Figure 21. Algebraic connectivity boundary equations and algebraic connectivity results when (a) capacity used increased on nodes of degree five from random modular graphs containing a minimum degree of three, and (b) capacity used increased on nodes of degree eight from random modular graphs containing minimum degree of three.

2. Accuracy of Congestion Estimation

Two congestion estimation simulations are presented to determine the accuracy of (36), from Chapter III, in estimating nodal congestion when a node crosses the congestion threshold established by a phantom node. The first simulation focuses on the accuracy of the method for a network containing one phantom node. The second simulation focuses on the accuracy of the method for a network containing multiple phantom nodes.

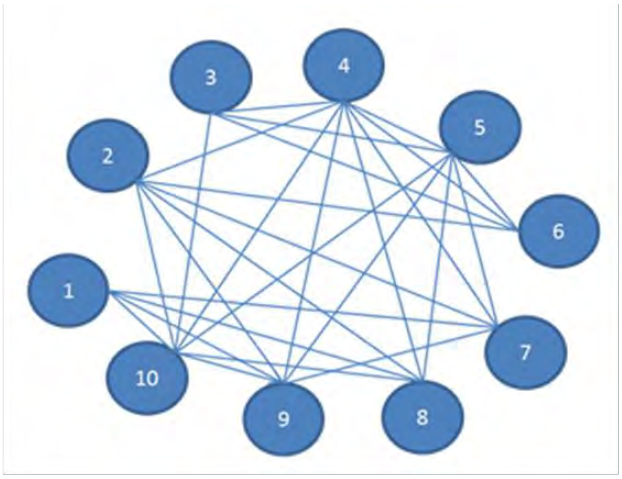
a. Single Phantom Node Case

The congestion estimation method sets the degree of the phantom node D_p as the threshold instead of the minimum degree $D_{\min}$ node in (29) to estimate a node's congestion when it crosses a congestion threshold. To demonstrate the accuracy of this method for a single congestion threshold, a network with a single phantom node was simulated to experience nodal congestion. This simulation was conducted for 20 different networks to ensure accuracy of the results. Since all of the simulations produced similar results, the results of a single network are presented here.

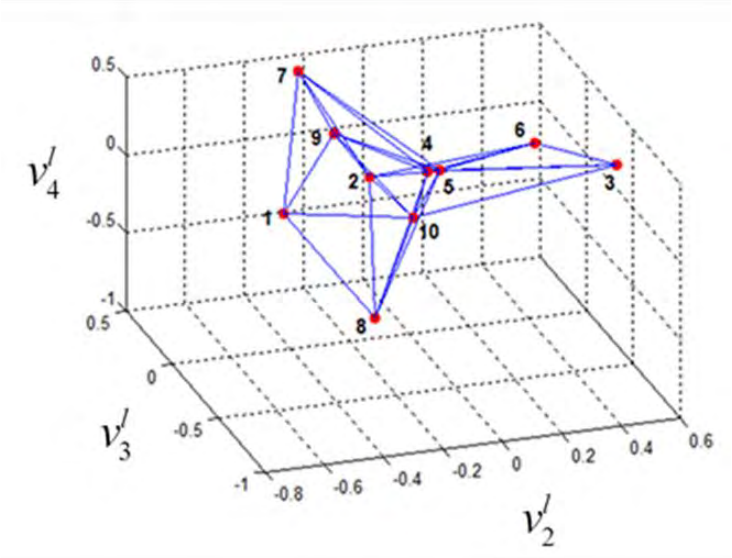
The inputs $n=10$, $c=2$, $p=0.5$, and $r=0.5$ [35] were used to form the adjacency matrix

$$A=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}. \tag{45}$$

The resultant network topology of (45) is shown in Figure 22(a). The network centrality is plotted using (v_2^l, v_3^l, v_4^l) [29], as shown in Figure 22(b), which were calculated from (45) using (5) and (8). The graph of network centrality was used to determine the initial placement of the phantom node. A single phantom node with $D_p = 1$ was initially attached to node 4 since it has $D_{\max}$ and is one of the most central nodes in the network.



(a) Network topology



(b) Network centrality

Figure 22. (a) Network topology of a random modular ten-node network with two modules and an overall probability of attachment of 0.5 (b) Network centrality graph of network using second, third, and fourth eigenindices, after [29].

For the simulation, the link weights for each node were incrementally decreased from one to zero. After each decrease, the eigenvector matrices V were recalculated from the new adjacency matrix A. The results were then compiled to monitor each node's health via nodal influence, and the point of congestion threshold crossing was determined. The node's capacity used at the intersection C_{Uis} was measured at the instant the node had $v_i^{l\max}$ of a smaller eigenindex i compared to the phantom node. Then the phantom node was relocated on a different node in the network, and the process was repeated until all nodes had been attached to the phantom node. Lastly, C_{Uis} was calculated using (36) from Chapter III. The results were compiled to determine the accuracy of the congestion estimation method.

The results of measured and estimated capacity used at the intersection C_{Uis} are shown in Table 3 for the network with the phantom node attached to node 4. Excluding node 4, we see that the maximum difference is 3.25 percent between the actual measured percent $C_{U\,\text{intersect}}$ and the estimated percent C_{Uis}, and the minimum difference is 0.21 percent. The average difference is 1.5 percent. These results validate the accuracy of the congestion estimation method for all nodes except the node attached to the phantom node in a single phantom node network. The difference between measured and estimated C_{Uis} for node 4 was much higher, which indicates the congestion estimation method is not accurate for a node attached to a phantom node.

Next, the results for when each node was attached to the phantom node were compiled, as shown in Table 4, to verify the inaccuracy of the estimation method for these types of nodes. The maximum difference is 18 percent and the minimum difference is 10 percent between the measured and estimated percent capacity used at the intersection C_{Uis}. The average difference is 14.14 percent. These differences are much higher than the measurements in Table 3. We associate these results to the Heisenberg uncertainty principle; the attachment of the phantom node to a node in the network for measurement purposes in turn affects the measurement of the node [37]. Therefore, the estimation method is accurate in estimating a node's congestion at the instant it crosses the congestion threshold for all nodes except the node attached to the phantom node.

Table 3. Percent difference between estimated and measured capacity used for nodes of a random ten node network containing a phantom with link weight equal to one attached to node 4.

Node	Node Degree (including Phantom node link)	Phantom Node Degree	Estimated C_{Uis} (percent)	Measured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
1	4	1	80	76.75	3.25
2	6	1	85.71	85.17	0.54
3	4	1	80	77	3.00
4	9 (Phantom $d=1$)	1	90	100	10.00
5	7	1	87.5	87.29	0.21
6	4	1	80	77	3.00
7	5	1	83.33	82	1.33
8	5	1	83.33	82	1.33
9	6	1	85.71	85.1	0.61
10	7	1	87.5	87.29	0.21
Average Difference (percent) *Including Node Attached to Phantom Node					2.348
Average Difference (percent) *Excluding Node Attached to Phantom Node					1.50

Table 4. Percent difference between estimated and measured capacity used for nodes of a random ten node network when the node is attached to phantom node with link weight equal to one.

Node	Total Node Degree (node degree plus phantom degree)	Phantom Node Degree	Estimated C_{Uis} (percent)	Meaured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
1	5	1	82	100	18
2	7	1	87.29	100	12.71
3	5	1	82	100	18
4	9	1	90	100	10
5	8	1	88.89	100	11.11

Node	Total Node Degree (node degree plus phantom degree)	Phantom Node Degree	Estimated C_{Uis} (percent)	Meaured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
6	5	1	82	100	18
7	6	1	85.1	100	14.9
8	6	1	85.1	100	14.9
9	7	1	87.29	100	12.71
10	8	1	88.89	100	11.11
Average Difference (percent)					14.14

The simulation results validate the estimation method for a network containing a single phantom node with a link weight equal to one and the need for phantom node relocation in the proposed monitoring scheme. The focus of the next simulation is on networks containing multiple phantom nodes and link weights not equal to one.

b. Multiple Phantom Node Case

Two types of networks were simulated for the multiple phantom node case: networks containing two phantom nodes and networks containing three phantom nodes. For each simulation, the phantom nodes had different link weight values, which created different congestion threshold levels within the dual basis. The link weights are non-integer values to ensure the primary nodal influence in a particular eigenindex. The networks were simulated to experience congestion to determine the accuracy of the estimation method for the different congestion thresholds.

For both simulations, the link weights for each node in the network were incrementally decreased from one to zero. After each decrement, the eigenvector matrix V was recalculated from the new adjacency matrix A, and the results were compiled to track the nodal influence of a node in the eigenindex i spectrum. The congestion threshold crossings were then identified, and the node's capacity used at the intersection

C_{Uis} was measured at each instance. Lastly, C_{Uis} was calculated using (36) from Chapter III and compared to the measured results to determine the accuracy of the estimation method.

(1) Two Congestion Thresholds

This simulation was conducted on 15 networks containing two phantom nodes, but since the simulation results were similar, only the results of a single network are presented here. The adjacency matrix A for the network is

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1.5 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0.5 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (46)$$

which resulted in the network topology shown in Figure 23. The green circles represent the phantom nodes and the green dashed lines represent links that exist in the adjacency matrix A but not in the actual physical network. The phantom node attached to node 8 has a link weight equal to 0.5, and the phantom node attached to node 3 has a link weight equal to 1.5.

The results of the simulation using (46) are shown in Table 5. Excluding the nodes attached to phantom nodes and nodes 11, 12, and 13, we see that the largest difference between measured and estimated percent capacity used at the intersection C_{Uis}

is 5.43 percent. Additionally, the average difference is 5.6 percent when the results from nodes attached to the phantom nodes are excluded. These results are slightly larger than the ones seen in the single phantom node case, but they still demonstrate that the estimation method is accurate for nodes that have a larger degree than the phantom node.

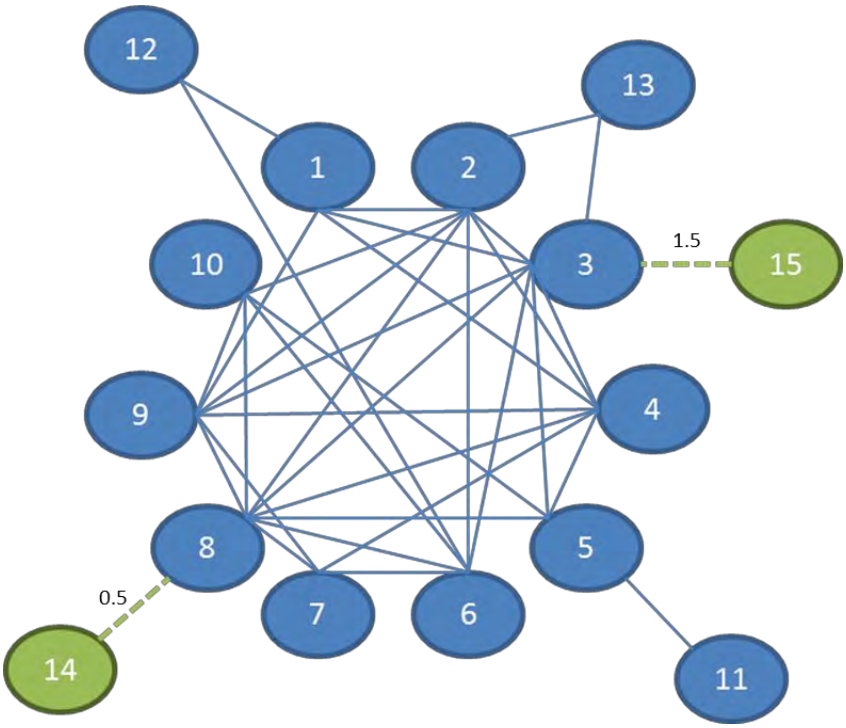


Figure 23. Network topology of 15 node network containing a phantom node (designated by green circle) with link weight equal to 0.5 attached to node 8 and a phantom node with link weight equal to 1.5 attached to node 3.

Table 5. Percent difference between estimated and measured capacity used for nodes of a random fifteen node network containing two phantom nodes, one with link weight equal to 0.5 attached to node 8 and one with link weight equal to 1.5 attached to node 3.

Node	Total Node Degree (node degree + phantom degree)	Phantom Node Degree	Estimated C_{Uis} (percent)	Measured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
1	5	0.5	91.67	90.2	1.47
		1.5	75	79.4	4.4
2	8	0.5	94.44	94.1	0.34
		1.5	83.33	84	0.67
3	9.5 (Phantom $d = 1.5$)	0.5	95.23	77.1	18.13
		1.5	85.71	100	14.29
4	7	0.5	93.75	93.2	0.55
		1.5	81.25	82.1	5.15
5	5	0.5	91.67	90.8	0.87
		1.5	75	78	3.00
6	6	0.5	92.86	92	0.86
		1.5	78.57	84	5.43
7	4	0.5	90	88	2.00
		1.5	70	67.5	2.50
8	9.5 (Phantom $d = 0.5$)	0.5	95.23	100	4.77
		1.5	85.71	83.6	2.11
9	7	0.5	93.75	93.2	0.55
		1.5	81.25	82.3	1.05
10	5	0.5	91.67	90.5	1.17
		1.5	75	73.8	1.2
11	1	0.5	75	45.5	29.5
		1.5	25	0	25
12	2	0.5	83.33	75	8.33
		1.5	50	30.5	19.5
13	3	0.5	87.5	84	3.5
		1.5	62.5	56.3	6.2
Average Difference (percent) *Including Nodes Attached to Phantom Node					6.25
Average Difference (percent) *Excluding Nodes Attached to Phantom Node					5.60

The results for nodes 11, 12, and 13 are not accurate. The maximum difference between measured and estimated percent capacity used at the intersection C_{Uis} occurs on node 11, but all three nodes have large differences. For each case, the estimation method overestimated the percent C_{Uis}, which means the node would be identified as congested well below the expected congestion level. This implies that the estimation method is either inaccurate for nodes that have small degrees or that the level of accuracy is dependent upon the difference between D_I and D_P. This is explored later in this chapter.

The simulation results show that the congestion estimation method accurately estimates the percent capacity used C_U on a node at the instant it crosses a congestion threshold. The results also show that by adding additional phantom nodes into the network, multiple congestion thresholds are produced. This means that nodes can be assigned to different congestion thresholds. For example, nodes 1, 2, and 3 of the network used in this simulation can be assigned to the congestion threshold formed by the phantom node with a link weight of 0.5, and the remaining nodes can be assigned to the congestion threshold formed by the other phantom node. This means that nodes 1, 2, and 3 are identified as congested in the range of 45 to 84 percent instead of 0 to 56 percent, and the remaining nodes are identified as congested in the range of 67 to 84 percent instead of 88 to 94 percent. The ability to accurately implement additional phantom nodes allows the monitoring scheme to be tailored to specific network requirements.

(2) Three Congestion Thresholds

The networks with three phantom nodes were formed by adding an additional node to the 15 networks simulated in the previous subsection. All of the results were similar in each of the 15 networks, so only one network's results are presented here.

$$A = \begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 2.5 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1.5 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0.5 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{47}$$

The adjacency matrix was formed by adding a third phantom node to (46). The third phantom node is attached to node 2 with a link weight of 2.5. The resultant network topology is shown in Figure 24. Again, green circles represent the phantom nodes, and the green dashed lines represent links that exist in the adjacency matrix A but not in the actual physical network.

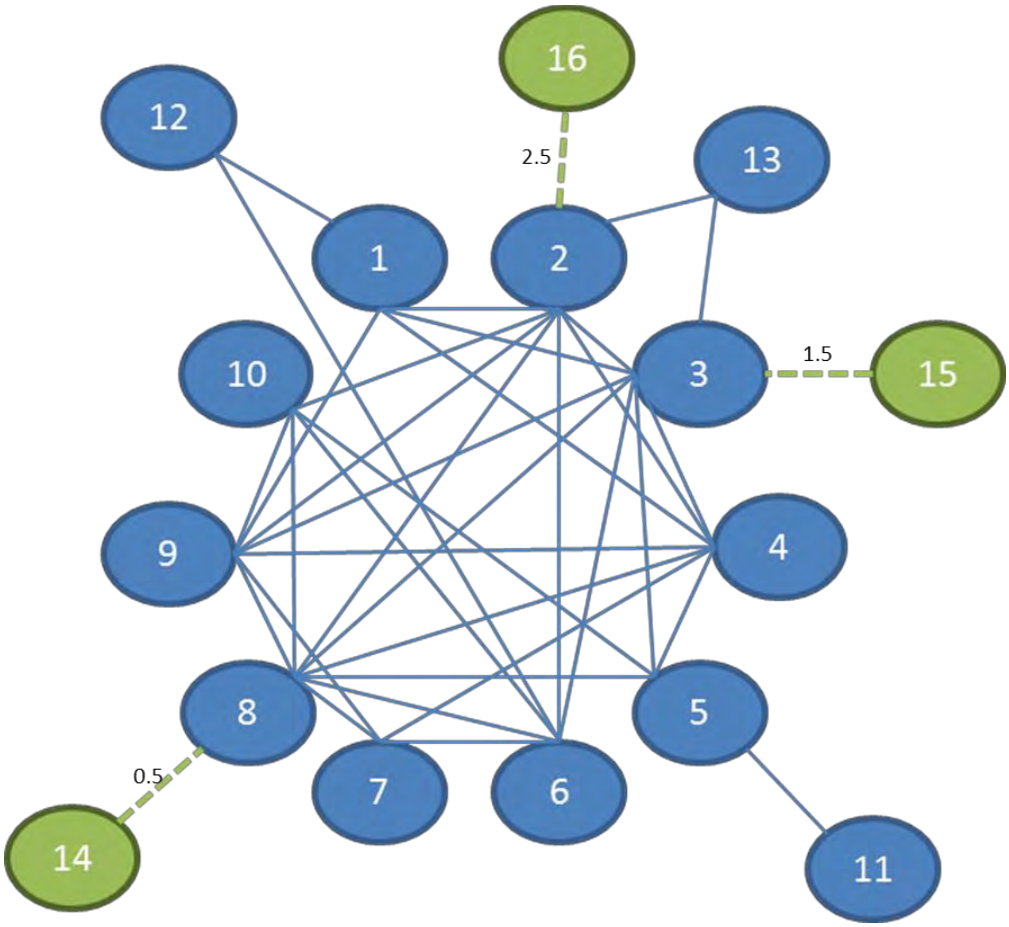


Figure 24. Network topology 16-node network containing a phantom node (designated by green circles) with link weight equal to 0.5 attached to node 8, a phantom node with link weight equal to 1.5 attached to node 3, and a phantom node with link weight equal to 2.5 attached to node 2.

The results of the measured and estimated capacity used at the intersection C_{Uis} are shown in Table 6 for all nodes in the network for each of the three congestion thresholds. The average difference between measured and estimated C_{Uis} is 5.1 percent when the results of the nodes attached to phantom nodes are excluded. This means that on average a node will be identified by the monitoring scheme as congested within 5.1 percent of the estimated C_{Uis} value.

Table 6. Percent difference between estimated and measured capacity used for nodes of a random ten node network containing three phantom nodes, one with link weight equal to 0.5 attached to node 8, one with link weight equal to 1.5 attached to node 3, and one with link weight equal to 2.5 attached to node 2.

Node	Total Node Degree (node + phantom degree)	Phantom Node Degree	Estimated C_{Uis} (percent)	Measured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
1	5	0.5	91.67	90.2	1.47
		1.5	75	75.8	0.8
		2.5	58.33	60.4	2.07
2	10.5 (Phantom $d = 2.5$)	0.5	95.65	79.3	16.35
		1.5	86.96	100	13.04
		2.5	78.26	100	21.74
3	9.5 (Phantom $d = 1.5$)	0.5	95.23	76.3	18.93
		1.5	85.71	100	14.29
		2.5	76.19	77.4	1.21
4	7	0.5	93.75	93.2	0.55
		1.5	81.25	82.1	0.85
		2.5	68.75	69	0.25
5	5	0.5	91.67	90.8	0.87
		1.5	75	75.4	0.40
		2.5	58.33	57.2	1.13
6	6	0.5	92.86	92	0.86
		1.5	78.57	78.8	0.23
		2.5	64.29	67.7	3.41
7	4	0.5	90	88	2.00
		1.5	70	67.5	2.50
		2.5	50	38.5	11.5
8	9.5 (Phantom $d = 0.5$)	0.5	95.23	100	4.77
		1.5	85.71	83.6	2.11
		2.5	76.19	76.5	0.31
9	7	0.5	93.75	93.2	0.55
		1.5	81.25	82.3	1.05
		2.5	68.75	70	1.25
10	5	0.5	91.67	90.5	1.17
		1.5	75	74.2	0.8
		2.5	58.33	55.8	2.53
11	1	0.5	75	45.5	29.5
		1.5	25	0	25
		2.5	0	0	0

Node	Total Node Degree (node + phantom degree)	Phantom Node Degree	Estimated C_{Uis} (percent)	Measured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
12	2	0.5	83.33	75	8.33
		1.5	50	29	21
		2.5	16.67	0	16.67
13	3	0.5	87.5	84	3.5
		1.5	62.5	57	5.5
		2.5	37.5	30.3	7.2
Average Difference (percent) *Including Nodes Attached to Phantom Node					6.3
Average Difference (percent) *Excluding Nodes Attached to Phantom Node					5.1

While the average difference shows the estimation method to be accurate, nodes 11, 12, and 13 still produced large differences between measured and estimated capacity used at the intersection C_{Uis}. This further supports the claim that the estimation method is either inaccurate for nodes that have small degrees or that the level of accuracy is dependent upon the difference between D_I and D_p. To determine the cause of the inaccuracy, the simulation was conducted again on the network shown in Figure 24, but the link weight of the phantom node attached to node 8 was changed from 0.5 to 0.1. The results are shown in Table 7.

Comparing the entries shown in Table 7 with the associated entries shown in Table 6, we see that the difference between estimated and measured capacity used at the intersection C_{Uis} decreased significantly by increasing the difference between the nodal degree of interest D_I and the phantom degree D_p; the equation for estimating C_{Uis} is more accurate when there is a large difference between D_I and D_p. This means that if a link weight value is selected for a phantom node that is similar to the degree of a node it is meant to monitor, then the node will be identified as congested at a smaller than expected C_U; a smaller C_U means that the controller will be more active because threshold crossing will occur more frequently and at a congestion level significantly below failure.

Table 7. Percent difference between estimated and measured capacity used for three low degree nodes in a random 13 node network containing a phantom node with a degree of 0.1 attached to node 8.

Node	Total Node Degree (node + phantom degree)	Phantom Node Degree	Estimated C_{Uis} (percent)	Measured C_{Uis} (percent)	Difference Between Estimated and Measured (percent)
11	1	0.1	95	90	5
12	2	0.1	96.67	95	1.67
13	3	0.1	97.5	96.8	0.7

The congestion estimation simulations validated the effectiveness of the method in determining the congestion level of a node when it crosses a congestion threshold. It also showed that different link weight values produce different congestion thresholds; the use of multiple congestion thresholds with different link weights allows the user to set a subset of the nodes to be monitored in comparison to a specific phantom node for the purpose of producing desired congestion identification. One significant result produced by the simulations was the inaccuracy of the estimation method for nodes attached to the phantom node and nodes that have a similar degree as the phantom node. This validates the need for phantom node relocation in the proposed monitoring scheme. It also informs the user that a large difference between phantom node degree and the node degrees of the set of nodes assigned to be monitored by the phantom node is required to ensure the accuracy of the monitoring scheme.

The simulations presented in this chapter validate the proposed SDN monitoring scheme described in Chapter III. The results showed that graph theory based methods allow the monitoring scheme to accurately identify disconnected, congested, underutilized, and attacked nodes through the use of phantom nodes. This means that the monitoring scheme is capable of producing the required outputs needed to update flow rules and manage the network. Additionally, the results showed that the monitoring scheme can be customized for individual networks. At network initialization, the user can use the congestion estimation method to determine the number of phantom nodes, the link weights of the phantom nodes, and which nodes are assigned to each phantom node

to ensure all nodes are identified as congested at an acceptable congestion level. Overall, the simulations and results of this chapter demonstrate the functionality and effectiveness of the proposed monitoring scheme.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

Spectral graph theory analysis was examined in this thesis to develop a network monitoring scheme to be implemented at the application layer of a SDN. The dual basis was identified as a tool to track node health in a dynamic network. The dual basis produces two metrics, nodal centrality and nodal influence, that provide information on the strength and connectivity of a node compared to other nodes in the network [17], [23]. Since these metrics change with respect to a change in a node's capacity available C_A, the concept of reference node was proposed and used in the monitoring scheme to identify changes in nodal conditions.

The concept of reference node uses phantom nodes in the adjacency matrix to establish congestion thresholds in the dual basis, which can be used to monitor changes in nodal influences. The location of the congestion threshold in the eigenindex i is dependent upon the link weight value of the phantom node; a link weight value is selected so that the phantom node has primary nodal influence $v_i^{l_{\max}}$ for a specific eigenindex i. As a node experiences an increase or a decrease in capacity available C_A due to traffic on its links [19], the nodal influence changes in the eigenindex i. The proposed monitoring scheme compares these changes to the capacity-static phantom node to produce outputs that identify congested, underutilized, and attacked nodes. These outputs can be implemented into the route generation function of the SDN controller to provide effective network management.

Additionally, the proposed network monitoring scheme was shown to be customizable through the use of multiple phantom nodes, which produce multiple congestion thresholds. The proposed congestion estimation method allows the user to estimate nodal congestion at a threshold crossing in order to select the appropriate link weight value for each phantom node to achieve desired monitoring results.

A. SIGNIFICANT RESULTS

The proposed monitoring scheme uses novel concepts and methods to identify nodal conditions and estimate congestion. The concept of using the dual basis to monitor changes in nodal connectivity over time and comparing these changes to a capacity-static phantom node to identify disconnections, congestion, underutilization, and attack has not been reported in literature to date. Additionally, the method of estimating nodal congestion using the degree of the node and the link weight of the phantom node has not previously been discussed in literature. These concepts and methods allow the proposed monitoring scheme to provide outputs to a SDN controller for the purpose of updating flow rules to provide timely network management.

The controller's ability to use the dual basis to monitor changes in nodal connectivity over time was demonstrated in the node health tracking simulation. The simulation showed that a node's centrality and nodal influence are directly tied to the capacity available on its links. The nodal influence of a node can be monitored to identify changes in connectivity relative to other nodes in the network. This allows strongly connected nodes to be distinguished from weakly connected nodes in a dynamic network.

The controller's ability to identify congestion, underutilization, and attack was demonstrated in the node health identification simulation. The implementation of a phantom node in the adjacency matrix establishes a congestion threshold in the dual basis that is used to identify changes in nodal conditions. It was shown that congested and attacked nodes are identified when they have primary nodal influence in a smaller eigenindex compared to the phantom node. Conversely, an underutilized node is identified when it has primary nodal influence in a larger eigenindex compared to the phantom node. The identification of these nodal conditions allows the SDN controller to identify changes that need to be made to flow rules to provide effective network management.

Finally, the method for estimating congestion was shown to be accurate for networks that contained either a single phantom node or multiple phantom nodes. The average difference between estimated and measured congestion was 1.5 percent for the

network with one phantom node, 5.6 percent for the network with two phantom nodes, and 5.1 percent for the network with three phantom nodes when excluding the results of the node attached to the phantom node. It was also shown that these average differences decrease as the difference between the degree of the phantom node and the degree of the node being monitored increases. The accuracy of this estimation method is significant because it allows a user to customize the proposed monitoring scheme to acceptable levels of identify nodal congestion.

B. RECOMMENDATIONS

The various requirements of each phase of the network monitoring scheme were described in Chapter III, but not all ideas and concepts were fully explored in this thesis. For example, a method to identify a nodal attack that is in progress was presented in the subsection for determining congestion, underutlization and attack. This method was not simulated in this thesis because MATLAB is not an effective simulation program for modeling continuous, dynamic traffic in a network. This method could be evaluated using a different program, such as OPNET, or measured using a SDN test bench.

The proposed monitoring scheme calls for additional SDN applications to be developed, specifically routing applications and security applications, to integrate the outputs of the monitoring scheme into the route generation function of the controller. Future efforts should focus on the development of an application that acts as an intrusion detection system for the network and an application that determines routing changes that need to be made based on network state.

All of the simulations in this thesis were limited by the inability to simulate and measure real-world network traffic. Future research should focus on implementing the proposed monitoring scheme in a SDN test bench to validate the results observed in this thesis. This requires the proposed monitoring scheme to be developed into an application that can be implemented in the application layer of the SDN test bench. This is the next major step towards implementing the proposed monitoring scheme in a large, complex network.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MATLAB RANDOM NETWORK GENERATION AND PLOTTING

The MATLAB code in this appendix was used in to create adjacency matrices of random modular networks in Chapter III and Chapter IV. The code also produces the additional required graph theory matrices and plots the eigencentrality basis using the second, third, and fourth eigenvectors. The code requires additional modular coding for the "random_modular_graph" and "gplot3" functions, which can be found in [35] and [29], respectively.

```
% Main Function - Create Adjacency Matrix
n=10; % n - number of nodes
c=2; % c - number of clusters/modules
p=.6; % p - overall probability of attachment
r=.5; % r - proportion of links within modules


[adj, modules] = random_modular_graph(n, c, p, r);
Adj = adj

% Determine Graph Theory Matrices
DegMat = diag(sum(Adj)); % create degree matrix
Lap = DegMat-Adj;      % create Laplacian matrix
[V,D] = eig(Lap);      % create eigenvalue and eigenvector matrices

% Plot Eigencentrality Basis
figure(1)
gplot3(Lap,[V(:,2),V(:,3),V(:,4)])
grid
hold on
set(gcf,'Color','white')
xlabel('Second Eigenvector')
ylabel('Third Eigenvector')
zlabel('Fourth Eigenvector')
title('Node Influence on Eigenvectors at 100% Network Capacity')
for x=1:n
  plot3(V(x,2),V(x,3),V(x,4),'.r','MarkerSize',20)
  str=sprintf('%0.0f',x);
  text(V(x,2)-0.08,V(x,3)-0.02,V(x,4),str);
end

for x2=1:n                  % cycle through rows
  links(x2) = sum(Adj(x2,:));       % determine number of links per
node
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATLAB ALGEBRAIC CONNECTIVITY BOUNDARIES AND LINK CONNECTIVITY CODE

The code presented in this appendix was used to generate the algebraic connectivity boundary equation results in Chapter IV. The code utilizes [35] to produce the random adjacency matrix. This code was then manipulated to produce the results in the link failure simulation in Chapter IV; the algebraic connectivity data for each degree node was averaged for all of the random networks and plotted on a single graph.

```
% Algebraic Connectivity vs. Capacity:

z2 = 30;  % # cycles through program (# graphs to be analyzed)
n4 = 0;     n5 = 0;     n6 = 0;

% Initialize matrices
AvgAC1 = zeros(z2,11);  AvgAC2 = zeros(z2,21);  AvgAC3 = zeros(z2,31);
AvgAC4 = zeros(z2,41);  AvgAC5 = zeros(z2,51);  AvgAC6 = zeros(z2,61);
AvgAC7 = zeros(z2,71);  AvgAC8 = zeros(z2,81);  AvgAC9 = zeros(z2,91);
AvgAC10 = zeros(z2,101);

% Initialize variables
b1 = 0; b2 = 0; b3 = 0; b4 = 0; b5 = 0; b6 = 0; b7 = 0; b8 = 0;
b9 = 0; b10 = 0;

for z=1:z2
%% Creating a modular network with no traffic

n=15; % n - number of nodes
c=3; % c - number of clusters/modules
p=.4; % p - overall probability of attachment
r=.5; % r - proportion of links within modules

% Laplacian and Algebraic Connectivity
[adj, modules] = random_modular_graph(n, c, p, r);
Adj = adj;
DegMat = diag(sum(Adj));
Lap = DegMat-Adj;
[V,D] = eig(Lap);
AlCon=D(2,2);

for x2=1:n              % cycle through rows
  links(x2) = sum(Adj(x2,:));    % determine number of links per node
end

MinDeg = min(links)

if MinDeg == 3
```

```
%% Gradual Congestion

%Initialize variables and matrices
Adj2 = Adj;
a1 = 0;
a2 = 0;
AlCon_All = zeros(15,150);

for x2=1:n                  % cycle through rows
  links(x2) = sum(Adj(x2,:));       % determine number of links per
node
end

for row=1:n
  for x=1:10                % to cycle through all weights
    weight = 0.1;
    for x2=1:n              % for column cycle
      if Adj2(row,x2) ~= 0      % looks for presence of link
        LW = Adj2(row,x2)-weight;  % change link weight to .1 less
        Adj2(row,x2) = LW;      % change link row value
        Adj2(x2,row) = LW;      % change link column value
        a1 = 1;            % flag - indicates link changed
      end
      if a1 == 1            % if link changed
        a2 = a2+1;          % matrix position variable
        DM2 = diag(sum(Adj2));    % Degree matrix
        LP2 = DM2 - Adj2;       % Laplacian matrix
        [V2,D2] = eig(LP2);     % Eigenvalues and Eigenvectors
        AlCon_All(row,a2) = D2(2,2);
      end
      a1 = 0;              % reset a1 flag
    end
  end
  a2 = 0;                  % reset position variable
  Adj2 = Adj;                % reset Adjacency matrix
end


% Initialize variables
A1 = 0; A2 = 0; A3 = 0; A4 = 0; A5 = 0; A6 = 0; A7 = 0; A8 = 0;
A9 = 0; A10 = 0;

AlCon_1 = 0; AlCon_2 = 0; AlCon_3 = 0; AlCon_4 = 0; AlCon_5 = 0;
AlCon_6 = 0; AlCon_7 = 0; AlCon_8 = 0; AlCon_9 = 0; AlCon_10 = 0;

% separate AlCon data according to # links on a node
for x=1:n
  if links(x) == 1
    A1 = A1+1;                    % position variable
    AlCon_1(A1,1:11) = [AlCon,AlCon_All(x,1:9),0]; % Organize Alg. Conn
data according to # links
    [rows1,columns1] = size(AlCon_1);        % determine number of
nodes that have that # of links
  elseif links(x) == 2
```

```
    A2 = A2+1;
    AlCon_2(A2,1:21) = [AlCon,AlCon_All(x,1:19),0];
    [rows2,columns2] = size(AlCon_2);
  elseif links(x) == 3
    A3 = A3+1;
    AlCon_3(A3,1:31) = [AlCon,AlCon_All(x,1:29),0];
    [rows3,columns3] = size(AlCon_3);
  elseif links(x) == 4
    A4 = A4+1;
    AlCon_4(A4,1:41) = [AlCon,AlCon_All(x,1:39),0];
    [rows4,columns4] = size(AlCon_4);
  elseif links(x) == 5
    A5 = A5+1;
    AlCon_5(A5,1:51) = [AlCon,AlCon_All(x,1:49),0];
    [rows5,columns5] = size(AlCon_5);
  elseif links(x) == 6
    A6 = A6+1;
    AlCon_6(A6,1:61) = [AlCon,AlCon_All(x,1:59),0];
    [rows6,columns6] = size(AlCon_6);
  elseif links(x) == 7
    A7 = A7+1;
    AlCon_7(A7,1:71) = [AlCon,AlCon_All(x,1:69),0];
    [rows7,columns7] = size(AlCon_7);
  elseif links(x) == 8
    A8 = A8+1;
    AlCon_8(A8,1:81) = [AlCon,AlCon_All(x,1:79),0];
    [rows8,columns8] = size(AlCon_8);
  elseif links(x) == 9
    A9 = A9+1;
    AlCon_9(A9,1:91) = [AlCon,AlCon_All(x,1:89),0];
    [rows9,columns9] = size(AlCon_9);
  elseif links(x) == 10
    A10 = A10+1;
    AlCon_10(A10,1:101) = [AlCon,AlCon_All(x,1:99),0];
    [rows10,columns10] = size(AlCon_10);
  end
end

% initialize spacing x value
ut1 = 0:(1/10):1;       ut2 = 0:(1/20):1;       ut3 = 0:(1/30):1;
ut4 = 0:(1/40):1;       ut5 = 0:(1/50):1;       ut6 = 0:(1/60):1;
ut7 = 0:(1/70):1;       ut8 = 0:(1/80):1;       ut9 = 0:(1/90):1;
ut10 = 0:(1/100):1;

if AlCon_3(1,1) ~= 0
  AvgAC3(z,1:31) = (sum(AlCon_3,1))/rows3;
  b3 = b3+1;
  for x=1:rows3
    AC3 = AlCon_3(x,:);
    figure(3)
    plot([0 1],[4 0],'r--')
    hold on
    plot([0 1],[3 3],'r:')
    plot(ut3,AC3)
    legend('a','b')
```

```
    title('Node with three Links')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    set(gcf,'Color','white')
  end
end

if AlCon_4(1,1) ~= 0
  AvgAC4(z,1:41) = (sum(AlCon_4,1))/rows4;
  b4 = b4+1;
  for x=1:rows4
    n4 = n4+1;
    AC4 = AlCon_4(x,:);
    figure(4)
    plot([0 1],[5 0],'r--')
    plot([0 1],[3 3],'r:')
    hold on
    plot(ut4,AC4)
    title('Node with Four Link')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    legend('y = (         )','location','northeast')
    set(gcf,'Color','white')
  end
end

if AlCon_5(1,1) ~= 0
  AvgAC5(z,1:51) = (sum(AlCon_5,1))/rows5;
  b5 = b5+1;
  for x=1:rows5
    n5 = n5+1;
    AC5 = AlCon_5(x,:);
    figure(5)
    plot([0 1],[6 0],'r--')
    plot([0 1],[3 3],'r:')
    hold on
    plot(ut5,AC5)
    title('Node with Five Links')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    legend('y = (Max Degree + 1)(1 - Cu)','location','northeast')
    set(gcf,'Color','white')
  end
end

if AlCon_6(1,1) ~= 0
  AvgAC6(z,1:61) = (sum(AlCon_6,1))/rows6;
  b6 = b6+1;
  for x=1:rows6
    n6 = n6+1;
    AC6 = AlCon_6(x,:);
    figure(6)
    plot([0 1],[7 0],'r--')
    plot([0 1],[3 3],'r:')
    hold on
```

```
    plot(ut6,AC6)
    title('Node with Six Links')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    legend('y = (Max Degree + 1)(1 - Cu)','location','northeast')
    set(gcf,'Color','white')
  end
end

if AlCon_7(1,1) ~= 0
  AvgAC7(z,1:71) = (sum(AlCon_7,1))/rows7;
  b7 = b7+1;
  for x=1:rows7
    AC7 = AlCon_7(x,:);
    figure(7)
    plot(ut7,AC7)
    hold on
    plot([0 1],[8 0],'r--')
    plot([0 1],[3 3],'r:')
    title('Node with Seven Links')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    set(gcf,'Color','white')
  end
end

if AlCon_8(1,1) ~= 0
  AvgAC8(z,1:81) = (sum(AlCon_8,1))/rows8;
  b8 = b8+1;
  for x=1:rows8
    AC8 = AlCon_8(x,:);
    figure(8)
    plot(ut8,AC8)
    hold on
    plot([0 1],[9 0],'r--')
    plot([0 1],[3 3],'r:')
    title('Node with Eight Links')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    set(gcf,'Color','white')
  end
end

if AlCon_9(1,1) ~= 0
  AvgAC9(z,1:91) = (sum(AlCon_9,1))/rows9;
  b9 = b9+1;
  for x=1:rows9
    AC9 = AlCon_9(x,:);
    figure(9)
    plot(ut9,AC9)
    hold on
    plot([0 1],[10 0],'r--')
    plot([0 1],[3 3],'r:')
    title('Node with Nine Links')
    xlabel('Capacity Used (   )')
```

```
    ylabel('Algebraic Connectivity')
    set(gcf,'Color','white')
  end
end

if AlCon_10(1,1) ~= 0
  AvgAC10(z,1:101) = (sum(AlCon_10,1))/rows10;
  b10 = b10+1;
  for x=1:rows10
    AC10 = AlCon_10(x,:);
    figure(10)
    plot(ut10,AC10)
    hold on
    plot([0 1],[11 0],'r--')
    plot([0 1],[3 3],'r:')
    title('Node with Ten Links')
    xlabel('Capacity Used (   )')
    ylabel('Algebraic Connectivity')
    set(gcf,'Color','white')
  end


end
end
end
```

APPENDIX C. MATLAB CONGESTION CODE

The code presented in this appendix was used to generate the node tracking results in Chapter IV. It utilizes the code found in [35] and [29]. This code was then manipulated to produce the node health identification results and congestion threshold accuracy results in Chapter IV. For node health identification, an additional single link node was placed in the created adjacency matrix to represent the phantom node. For congestion threshold accuracy results, additional phantom nodes were added to the adjacency matrix and the plots were used to provide measurement values.

```
% Tracking Node Health Simulation
row = 1   % Node that is going to experience change in congestion

n=8; % n - number of nodes
c=2; % c - number of clusters/modules
p=.45; % p - overall probability of attachment
r=.3; % r - proportion of links within modules

% Create Adjacecny Matrix
%[adj, modules] = random_modular_graph(n, c, p, r);
%Adj = adj

DegMat = diag(sum(Adj)); % create degree matrix
Lap = DegMat-Adj;      % create Laplacian matrix
[V,D] = eig(Lap);      % create eigenvalue and eigenvector matrices
AlCon=D(2,2);        % algebraic connectivity

for x2=1:n               % cycle through rows
  links(x2) = sum(Adj(x2,:));    % determine number of links per node
end

% output max, min and number of links
links
MinDeg = min(links)
MaxDeg = max(links)
degree = links(row);  % degree of node that experiences congestion

figure(1)
gplot3(Lap,[V(:,2),V(:,3),V(:,4)])
grid
hold on
set(gcf,'Color','white')
xlabel('Second Eigenvector')
ylabel('Third Eigenvector')
zlabel('Fourth Eigenvector')
```

```
title('Node Influence on Eigenvectors at 100% Network Capacity')
for x=1:n
  plot3(V(x,2),V(x,3),V(x,4),'.r','MarkerSize',20)
  str=sprintf('%0.0f',x);
  text(V(x,2)-0.08,V(x,3)-0.02,V(x,4),str);
end

%% Gradual Increase of Capacity Used on Max Degree Node

Adj2 = Adj;   % to be able to access original Adj matrix
weight = 0.01; % increment of congestion
a1 = 0;     % flag for change in congestion
a2 = 0;     % matrix position variable
a3 = 0;     % matrix position variable
b = degree*100; % degree x 100 congestion values (using 0.01)

% initialize matrices for storage of data
Lambda2 = zeros(1,b);
Lambda3 = zeros(1,b);
Lambda4 = zeros(1,b);
Lambda5 = zeros(1,b);
Lambda6 = zeros(1,b);
Lambda7 = zeros(1,b);
Lambda8 = zeros(1,b);

for x=1:100               % cycle through all weights
  for x2=1:n              % cycle through the columns
    if Adj2(row,x2) ~= 0      % looks for presence of link
      LW = Adj2(row,x2)-weight;  % change link weight to .1 less
      Adj2(row,x2) = LW;      % change link row value
      Adj2(x2,row) = LW;      % change link column value
      a1 = 1;           % set link change flag
    end
    if a1 == 1            % if link changed
      a2 = a2+1;          % variable used for matrix position
      DM2 = diag(sum(Adj2));   % Degree matrix
      LP2 = DM2 - Adj2;       % Laplacian matrix
      [V2,D2] = eig(LP2);      % Eigenvalues and Eigenvectors
      Lambda2(1,a2) = D2(2,2); % Puts Lambda 2 value into next position
in matrix
      Lambda3(1,a2) = D2(3,3);       Lambda4(1,a2) = D2(4,4);
      Lambda5(1,a2) = D2(5,5);       Lambda6(1,a2) = D2(6,6);
      Lambda7(1,a2) = D2(7,7);       Lambda8(1,a2) = D2(8,8);
      a3 = a3+1;          % variable used for matrix position
      Fiedler_V2(:,a3) = V2(:,2); % puts eig-vectors into next position
in matrix
      Fiedler_V3(:,a3) = V2(:,3);    Fiedler_V4(:,a3) = V2(:,4);
      Fiedler_V5(:,a3) = V2(:,5);    Fiedler_V6(:,a3) = V2(:,6);
      Fiedler_V7(:,a3) = V2(:,7);    Fiedler_V8(:,a3) = V2(:,7);
    end
    a1 = 0; % set flag back to zero
  end
end
```

```
% square magnitudes of eigenvectors
Mag_FiedlerV2 = (Fiedler_V2).^2;    Mag_FiedlerV3 = (Fiedler_V3).^2;
Mag_FiedlerV4 = (Fiedler_V4).^2;    Mag_FiedlerV5 = (Fiedler_V5).^2;
Mag_FiedlerV6 = (Fiedler_V6).^2;    Mag_FiedlerV7 = (Fiedler_V7).^2;
Mag_FiedlerV8 = (Fiedler_V8).^2;

% Second Eigenvector
N1_V2 = Mag_FiedlerV2(1,:); N2_V2 = Mag_FiedlerV2(2,:);
N3_V2 = Mag_FiedlerV2(3,:); N4_V2 = Mag_FiedlerV2(4,:);
N5_V2 = Mag_FiedlerV2(5,:); N6_V2 = Mag_FiedlerV2(6,:);
N7_V2 = Mag_FiedlerV2(7,:); N8_V2 = Mag_FiedlerV2(8,:);

% Third Eigenvector
N1_V3 = Mag_FiedlerV3(1,:); N2_V3 = Mag_FiedlerV3(2,:);
N3_V3 = Mag_FiedlerV3(3,:); N4_V3 = Mag_FiedlerV3(4,:);
N5_V3 = Mag_FiedlerV3(5,:); N6_V3 = Mag_FiedlerV3(6,:);
N7_V3 = Mag_FiedlerV3(7,:); N8_V3 = Mag_FiedlerV3(8,:);

% Fourth Eigenvector
N1_V4 = Mag_FiedlerV4(1,:); N2_V4 = Mag_FiedlerV4(2,:);
N3_V4 = Mag_FiedlerV4(3,:); N4_V4 = Mag_FiedlerV4(4,:);
N5_V4 = Mag_FiedlerV4(5,:); N6_V4 = Mag_FiedlerV4(6,:);
N7_V4 = Mag_FiedlerV4(7,:); N8_V4 = Mag_FiedlerV4(8,:);

% Fifth Eigenvector
N1_V5 = Mag_FiedlerV5(1,:); N2_V5 = Mag_FiedlerV5(2,:);
N3_V5 = Mag_FiedlerV5(3,:); N4_V5 = Mag_FiedlerV5(4,:);
N5_V5 = Mag_FiedlerV5(5,:); N6_V5 = Mag_FiedlerV5(6,:);
N7_V5 = Mag_FiedlerV5(7,:); N8_V5 = Mag_FiedlerV5(8,:);

% Sixth Eigenvector
N1_V6 = Mag_FiedlerV6(1,:); N2_V6 = Mag_FiedlerV6(2,:);
N3_V6 = Mag_FiedlerV6(3,:); N4_V6 = Mag_FiedlerV6(4,:);
N5_V6 = Mag_FiedlerV6(5,:); N6_V6 = Mag_FiedlerV6(6,:);
N7_V6 = Mag_FiedlerV6(7,:); N8_V6 = Mag_FiedlerV6(8,:);

% Seventh Eigenvector
N1_V7 = Mag_FiedlerV7(1,:); N2_V7 = Mag_FiedlerV7(2,:);
N3_V7 = Mag_FiedlerV7(3,:); N4_V7 = Mag_FiedlerV7(4,:);
N5_V7 = Mag_FiedlerV7(5,:); N6_V7 = Mag_FiedlerV7(6,:);
N7_V7 = Mag_FiedlerV7(7,:); N8_V7 = Mag_FiedlerV7(8,:);

% Eigth Eigenvector
N1_V8 = Mag_FiedlerV8(1,:); N2_V8 = Mag_FiedlerV8(2,:);
N3_V8 = Mag_FiedlerV8(3,:); N4_V8 = Mag_FiedlerV8(4,:);
N5_V8 = Mag_FiedlerV8(5,:); N6_V8 = Mag_FiedlerV8(6,:);
N7_V8 = Mag_FiedlerV8(7,:); N8_V8 = Mag_FiedlerV8(8,:);

% Initial Eigenvector Spectrum (8 node case)
I_N1 = [Mag_FiedlerV2(1,1), Mag_FiedlerV3(1,1), Mag_FiedlerV4(1,1),
Mag_FiedlerV5(1,1), Mag_FiedlerV6(1,1), Mag_FiedlerV7(1,1),
Mag_FiedlerV8(1,1)];
```

```
I_N2 = [Mag_FiedlerV2(2,1), Mag_FiedlerV3(2,1), Mag_FiedlerV4(2,1),
Mag_FiedlerV5(2,1), Mag_FiedlerV6(2,1), Mag_FiedlerV7(2,1),
Mag_FiedlerV8(2,1)];
I_N3 = [Mag_FiedlerV2(3,1), Mag_FiedlerV3(3,1), Mag_FiedlerV4(3,1),
Mag_FiedlerV5(3,1), Mag_FiedlerV6(3,1), Mag_FiedlerV7(3,1),
Mag_FiedlerV8(3,1)];
I_N4 = [Mag_FiedlerV2(4,1), Mag_FiedlerV3(4,1), Mag_FiedlerV4(4,1),
Mag_FiedlerV5(4,1), Mag_FiedlerV6(4,1), Mag_FiedlerV7(4,1),
Mag_FiedlerV8(4,1)];
I_N5 = [Mag_FiedlerV2(5,1), Mag_FiedlerV3(5,1), Mag_FiedlerV4(5,1),
Mag_FiedlerV5(5,1), Mag_FiedlerV6(5,1), Mag_FiedlerV7(5,1),
Mag_FiedlerV8(5,1)];
I_N6 = [Mag_FiedlerV2(6,1), Mag_FiedlerV3(6,1), Mag_FiedlerV4(6,1),
Mag_FiedlerV5(6,1), Mag_FiedlerV6(6,1), Mag_FiedlerV7(6,1),
Mag_FiedlerV8(6,1)];
I_N7 = [Mag_FiedlerV2(7,1), Mag_FiedlerV3(7,1), Mag_FiedlerV4(7,1),
Mag_FiedlerV5(7,1), Mag_FiedlerV6(7,1), Mag_FiedlerV7(7,1),
Mag_FiedlerV8(7,1)];
I_N8 = [Mag_FiedlerV2(8,1), Mag_FiedlerV3(8,1), Mag_FiedlerV4(8,1),
Mag_FiedlerV5(8,1), Mag_FiedlerV6(8,1), Mag_FiedlerV7(8,1),
Mag_FiedlerV8(8,1)];

% Initialize X-axis (Which Eigenvector)
EV = [2:1:n];
figure(2)
bar(EV,[I_N1', I_N2' I_N3' I_N4' I_N5' I_N6' I_N7' I_N8'])
hold on
ylim([0 1.4])
xlim([1.5 8.5])
xlabel('Eigenvalue index')
ylabel('Eigencentrality Metric')
legend('Node 1 (Degree 6)','Node 2 (Degree 3)','Node 3 (Degree
2)','Node 4 (Degree 4)','Node 5 (Degree 5)','Node 6 (Degree 5)','Node 7
(Degree 5)','Node 8 (Degree 4)','location','northeast')
set(gcf,'Color','white')
hold off

% Capacity Used Indexing
Cu = 0:(1/b):1;
Cu = Cu(2:(b+1));
Cu = Cu.*2;

figure(4)
subplot(2,1,2)
plot(Cu,N1_V8,'b')
hold on
plot(Cu,N2_V8,'k')      plot(Cu,N3_V8,'m')      plot(Cu,N4_V8,'c')
plot(Cu,N5_V8,'y')      plot(Cu,N6_V8,'r')      plot(Cu,N7_V8,'g')
plot(Cu,N8_V8,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(a) 8^t^h Eigenvector')
set(gcf,'Color','white')
hold off

figure(5)
```

```
subplot(2,1,1)
plot(Cu,N1_V7,'b')
hold on
plot(Cu,N2_V7,'k')      plot(Cu,N3_V7,'m')      plot(Cu,N4_V7,'c')
plot(Cu,N5_V7,'y')      plot(Cu,N6_V7,'r')      plot(Cu,N7_V7,'g')
plot(Cu,N8_V7,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(b) 7^t^h Eigenvector')
set(gcf,'Color','white')
hold off
subplot(2,1,2)
plot(Cu,N1_V6,'b')
hold on
plot(Cu,N2_V6,'k')      plot(Cu,N3_V6,'m')      plot(Cu,N4_V6,'c')
plot(Cu,N5_V6,'y')      plot(Cu,N6_V6,'r')      plot(Cu,N7_V6,'g')
plot(Cu,N8_V6,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(c) 6^t^h Eigenvector')
set(gcf,'Color','white')
hold off

figure(6)
subplot(2,1,1)
plot(Cu,N1_V5,'b')
hold on
plot(Cu,N2_V5,'k')      plot(Cu,N3_V5,'m')      plot(Cu,N4_V5,'c')
plot(Cu,N5_V5,'y')      plot(Cu,N6_V5,'r')      plot(Cu,N7_V5,'g')
plot(Cu,N8_V5,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(d) 5^t^h Eigenvector')
set(gcf,'Color','white')
hold off
subplot(2,1,2)
plot(Cu,N1_V4,'b')
hold on
plot(Cu,N2_V4,'k')      plot(Cu,N3_V4,'m')      plot(Cu,N4_V4,'c')
plot(Cu,N5_V4,'y')      plot(Cu,N6_V4,'r')      plot(Cu,N7_V4,'g')
plot(Cu,N8_V4,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(e) 4^t^h Eigenvector')
set(gcf,'Color','white')
hold off

figure(7)
subplot(2,1,1)
plot(Cu,N1_V3,'b')
hold on
plot(Cu,N2_V3,'k')      plot(Cu,N3_V3,'m')      plot(Cu,N4_V3,'c')
plot(Cu,N5_V3,'y')      plot(Cu,N6_V3,'r')      plot(Cu,N7_V3,'g')
plot(Cu,N8_V3,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(f) 3^r^d Eigenvector')
set(gcf,'Color','white')
legend('Node 1','Node 2','Node 3','Node 4','Node 5','Node 6','Node
7','Node 8','location','northeast')
```

```
hold off
subplot(2,1,2)
plot(Cu,N1_V2,'b')
hold on
plot(Cu,N2_V2,'k')      plot(Cu,N3_V2,'m')      plot(Cu,N4_V2,'c')
plot(Cu,N5_V2,'y')      plot(Cu,N6_V2,'r')      plot(Cu,N7_V2,'g')
plot(Cu,N8_V2,'b:')
ylim([0 1])             xlabel('Time (s)')      ylabel('Magnitude')
title('(g) 2^n^d Eigenvector')
set(gcf,'Color','white')
hold off
```

LIST OF REFERENCES

[1] A. Gelberger, Y. Yemini, and R. Giladi, "Performance analysis of software-defined networking (SDN)," in *Proc. of IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2013.

[2] D. L. Tennenhouse and D. J. Wetherall, "Towards and active network architecture," *SIGCOMM Computer Communication Review,* vol. 26, no. 2, pp. 5–17, 1996.

[3] M. Jarschel, T. Zinner, T. Hohn, and P. Tran-Gia, "On the Accuracy of Leveraging SDN for Passive Network Measurements," in *Proc. of Telecommunication Networks and Applications Conference (ATNAC)*, pp. 41–46, 2013.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] Open Networking Foundation. (2014). Open Networking Foundation homepage. [Online]. Available: https://www.opennetworking.org/.

[6] K. Bakshi, "Considerations for software defined networking (SDN): Approaches and use cases," in *Proc. of IEEE Aerospace Conference*, 2013.

[7] Open Networking Foundation. (2014). Solution brief: Operator network monetization through Openflow-enabled SDN. [Online]. Available: https://www.opennetworking.org/solution-brief-operator-network-monetization-through-openflow-enabled-sdn.

[8] OpenFlow Switch Consortium, "OpenFlow switch specification version 1.3.4," Open Networking Foundation, Palo Alto, 2014.

[9] Z. Bozakov and A. Rizk, "Taming SDN controllers in heterogeneous hardware environments," in *Proc. of Second European Workshop on Software Defined Networks*, 2013.

[10] H. Hata, "A study of requirements for SDN switch platform," in *Proc. of International Symposium on Intelligent Signal Processing and Communication Systems*, 2013.

[11] S. Huang and J. Griffioen, "Network hypervisors: Managing the emerging SDN chaos," in *Proc. of 22nd International Conference on Computer Communications and Networks*, 2013.

[12] S. Raza and D. Lenrow. (2013, October 6). Open Networking Foundation North Bound Interface Working Group charter. [Online]. Available: https://www.opennetworking.org/working-groups/northbound-interfaces.

[13] A. Jamakovic and S. Uhlig, "On the relationship between the algebraic connectivity and graph's robustness to node and link failures," in *Proc. of 3rd IEEE EURO-NGI Conference on Next Generation Internet Networks*, 2007.

[14] A. Jamakovic and S. Uhlig, "Influence of the network structure on robustness," in *Proc. of 15th IEEE International Conference on Networks (ICON)*, 2007.

[15] A. Sydney, "The evaluation of software defined networking for communication and control of cyber physical systems," Ph.D. diss. abstract, Dept. Elect. and Comput. Eng., Kansas State University, Manhattan, KA, 2013.

[16] H. Wang and P. V. Mieghem, "Algebraic connectivity optimization via link addition," in *Proc. of IEEE/ACM Bionetics*, 2008.

[17] P. V. Mieghem. (2014, Jan 18). "Graph eigenvectors, fundamental weights and centrality metrics for nodes in networks. Cornell University Library. [Online]. Available: arXiv preprint arXic:1401.4580[math.SP].

[18] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized? State distribution trade-offs in software defined networks," in *Proc. of HotSDN*, Helsinki, Finland, 2012.

[19] W. Stallings, *Data and Computer Communications*. Upper Saddle River, NJ: Prentice Hall, 2011.

[20] M. McDowell. (2013, February 6). Security tip (ST04-015): Understanding denial of service attacks. [Online]. Available: https://www.us-cert.gov/ncas/tips/ST04-015.

[21] P. V. Mieghem, *Graph Spectra for Complex Networks*. New York: Cambridge University Press, 2011.

[22] V. K. Balakrishnan, *Schaum's Outlines of Theory and Problems of Graph Theory*. New York: McGraw Hill, 1997.

[23] T. Parker, J. Johnson, M. Tummala, J. McEachen, and J. Scrofani, "Dynamic State Determination of a Software-Defined Network via Dual Basis Representation," in *Proc. of International Conference on Signal Processing and Communication Systems*, submitted for publication.

[24] V. Garg, *Wireless Communications and Networking*. San Francisco: Morgan Kaufmann, 2007.

[25] T. Parker, "Software Defined Networking Architecture and Traffic Engineering for Robustness and Security," Ph.D. diss.proposal, Dept. Elect. and Comput. Eng., Naval Postgraduate School, Monterey, CA, 2014.

[26] F. R. K. Chan. *Lectures on Spectral Graph Theory*. [Online]. Available: http://yaroslavvb.com/papers/chung-lectures.pdf.

[27] T. Parker, J. Johnson, M. Tummala, J. McEachen, and J. Scrofani, "Identifying Congestion in Software-Defined Networks," in *Proc. of 48th Asilomar Conference on Signals, Systems and Computers*, to be published.

[28] G. Strang, *Linear Algebra and Its Applications*. Belmont, CA: Brooks/Cole, Cengage Learning, 2006.

[29] A. M. Razeghi. (2008, April 9). *gplot3* [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/19521-gplot3.

[30] D. Spielman, "Spectral Graph Theory and Its Applications," in *Proc. of 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007.

[31] M. Fiedler, "Algebraic Connectivity of Graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298–305, 1973.

[32] Flowgrammable. (2014). Echo Request and Echo Response. [Online]. Available: http://flowgrammable.org/sdn/openflow/message-layer/echo/.

[33] C. Estan, "Automatically Inferring Patterns of Resource Consumption in Network Traffic," in *Proc. of SIGCOMM*, 2003.

[34] Cisco. (2014). *Snort Homepage* [Online]. Available: https://www.snort.org/.

[35] MIT Strategic Engineering. (2011). Matlab Tools for Network Analysis (2006–2011). [Online]. Available: http://strategic.mit.edu/downloads.php?page=matlab_networks.

[36] Mathworks Documentation Center. (2014). eig. [Online]. Available: http://www.mathworks.com/help/matlab/ref/eig.html.

[37] University of Oregon. (2014). Uncertainty principle. [Online]. Available: http://abyss.uoregon.edu/~js/21st_century_science/lectures/lec14.html.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California